P800850/DE/1

(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENTAMT

(12) Übersetzung der **europäischen Patentschrift**

(87) EP 0 502 975 B1

(10) **DE 690 31 078 T 2**

(51) Int. Cl.[6]:
G 06 F 17/50
G 06 F 19/00
G 06 F 9/45
G 06 F 9/44

DE 690 31 078 T 2

(21) Deutsches Aktenzeichen: 690 31 078.1
(86) PCT-Aktenzeichen: PCT/US90/07013
(86) Europäisches Aktenzeichen: 91 901 023.1
(87) PCT-Veröffentlichungs-Nr.: WO 91/08543
(86) PCT-Anmeldetag: 30. 11. 90
(87) Veröffentlichungstag der PCT-Anmeldung: 13. 6. 91
(87) Erstveröffentlichung durch das EPA: 16. 9. 92
(87) Veröffentlichungstag der Patenterteilung beim EPA: 16. 7. 97
(47) Veröffentlichungstag im Patentblatt: 15. 1. 98

(30) Unionspriorität:
444060 30.11.89 US

(73) Patentinhaber:
Seer Technologies, Inc., Cary, N.C., US

(74) Vertreter:
Rehberg und Kollegen, 37085 Göttingen

(84) Benannte Vertragstaaten:
AT, BE, CH, DE, DK, ES, FR, GB, IT, LI, LU, NL, SE

(72) Erfinder:
WADHWA, Vivek, K., Paramus, NJ 07652, US; ATAIE, Faraz, Brooklyn, NY 11215, US; AUBRUN, Vincent, P., New York, NY 10023, US; ERLIKH, Leonide, Brooklyn, NY 11224, US; FISCHER, Michael, Passaic, NJ 07055, US; FOCHLER, Michael, New York, NY 10026, US; HAYMAN, Craig, B., New York, NY 10280, US; HILDEBRAND, Daniel, Stamford, CT 06902, US; HUGHES, James, Hartsdale, NY 10530, US; LAMBERT, Jeffrey, L., East Brunswick, NJ 08816, US; LEE, Douglas, E., White Plains, NY 10603, US; LIM, Nicholas, R., London N7 0PU, GB; MODI, Rajan, S., New York, NY 10025, US; MOSEBACH, Richard, W., Hicksville, NY 11801, US; MOSKOWITZ, Joel, M., New York, NY 10021, US; OLOWU, Tayo, New York, NY 10016, US; POWER, Elaine, C., New York, NY 10003, US; SHING, Norman, New Hyde Park, NY 11040, US

(54) RECHNERUNTERSTÜTZTE SOFTWAREENTWICKLUNGSEINRICHTUNG

DE 690 31 078 T 2

Application No.: 91 901 023.1
Applicant: Seer Technologies, Inc.
8000 Regency Parkway
US - Cary, North Carolina 27511

RECHNERGESTÜTZTE SOFTWAREENTWICKLUNGSEINRICHTUNG

Gebiet der Erfindung

Die Erfindung bezieht sich auf ein Multiprozessorcomputersystem und insbesondere auf Einrichtungen, die bei der Entwicklung von Computerprogrammen für solche Systeme helfen.

Hintergrund der Erfindung

Computerunterstützte Softwareentwicklungssoftware (CASE) bzw. Softwareentwicklungseinrichtungen unterstützen Computerprogrammierer oder Systementwickler beim Planen, Entwickeln und Testen eines Computerprogramms.

Traditionell wurden die Schritte, die zum Umsetzen eines Computerprogramms notwendig sind, manuell ausgeführt. Entwicklungsteams folgten einer Zahl von diskreten Schritten, um ein brauchbares Anwendungsprogramm von einer ursprünglichen Idee aus zu entwickeln.

Die Analyse beginnt mit dem manuellen Entwickeln eines Modells, in dem ein Problem von einem Computer gelöst werden kann. Entwicklungsteams ermitteln die Bedürfnisse des zukünftigen Benutzers und die Eigenschaften, die das Computersystem besitzen sollte, um diese Anforderung zu erfüllen.

In der technischen Planungsphase der Entwicklung beginnen Entwickler damit, zu definieren, wie das Anwendungsprogramm auf einem gegebenen System aufgebaut wird. Sie bestimmen manuell die benötigten Verfahrens- und Datenelemente, und wie die Daten und

Verfahren zusammengesetzt werden, um die Softwarelösung auszubilden. In dieser Phase sind die Datenmodellbildung und die Prozeßmodellbildungdie beiden Hauptaufgaben.

Mit dem Basisplan des modellierten Programms beginnen Entwickler dann die Aufgabe der Eingabe des Codes des Programms. Computerprogramme werden im allgemeinen in höheren Programmiersprachen, wie beispielsweise BASIC, C, COBOL, FORTRAN oder PL/I geschrieben. Die Aufgabe, die theoretischen Planung einer Anwendung auf einen Arbeitscode zu reduzieren, ist eine mühsame Aufgabe, die viele Mannstunden erfordert. Erfahrung in der zur Entwicklung des Programms verwendeten Programmiersprache ist erforderlich.

Mit dem geschriebenen Code ist es das nächste Problem des Entwicklungsteams, das Programm hinsichtlich Syntaxfehlern zu entstören, und dann das Programm zu testen, um festzustellen, ob die Anwendung die gewünschten Funktionen ausführt. Typischerweise erfordert die Entstör- und Testphase, daß der Programmierer das Programm auf der Codeebene auswertet.

Die Entwicklungsaufgaben des Programmierers werden anspruchsvoller mit der Verwendung einer Architektur von Multiprocessingsystemen.

Traditionell gab es zwei grundlegende Hardwarekonfigurationen, die beim Aufbau von Computersystemen für eine Mehrzahl von Verwendern eingesetzt wurden. In einer Konfiguration wird die gesamte Verarbeitung des Systems von einem großen "Mainframe"-Computer ausgeführt. Jeder Benutzer greift auf dieses System über nicht-verarbeitende Endgeräte zu.

Ein Netzwerksystem ist die zweite traditionelle Hardwarekonfiguration. Ein Netzwerk besteht aus einer Anzahl von einzelnen verarbeitenden Einheiten, die zusammengeschaltet sind, um das gemeinsame Verwenden von Daten und Software zu ermöglichen. Dabei kann es zusätzliche Prozessoren geben, um die zentralisierte Datenbasis einer Gruppe zu pflegen, und zusätzliche

Prozessoren können der Aufrechterhaltung des Betriebs des Systems zugeordnet sein. Die Verwendung eines Netzwerks von Prozessoren, die jeweils einem spezifischen Aspekt des Computersystems zugeordnet sind, ist die Essenz des Multiprocessings.

Das Zunehmen der Verwendung von Multiprocessingsystemen kann auf die starke Verbreitung und Entwicklung von leistungsfähigen "Mikro-" bzw. "Personal-"Computern (PC's) zurückgeführt werden. In gewissen Grenzen kann ein PC viele derselben Verarbeitungsaufgaben ausführen, die Mainframe- bzw. Minicomputer ausführen. Ein PC kann diese Aufgaben jedoch mit einer erheblichen Einsparung an Anweisungen ausführen, die von dem Computer ausgeführt und in Millionen Anweisungen pro Sekunde, "MIPS", gemessen werden. Weiterhin ist ein PC ungleich einem Mainframe-System einem einzelnen Benutzer zugeordnet. Ein effizientes Multiprocessingsystem fördert die gemeinsame Verwendung von Daten und die Verwendung von bestimmten PC's für so viele Aufgaben wie möglich.

Bei Multiprozessorsystemen kann eine Anwendung auf mehr als einem Prozessor ausgeführt werden. Wenn verschiedene Teile eines Anwendungsprogramms auf getrennten Prozessoren ausgeführt werden, ist die Anwendung "verteilt". Verteilte Verarbeitung kann entweder in einer seriellen Sequenz oder parallel ausgeführt werden.

Die gleichzeitige Ausführung eines Programms auf vielen Prozessoren ist parallele Verarbeitung. Die sequentielle Ausführung eines Programms über verschiedene Hardwareumgebungen ist serielle bzw. "kooperative" Verarbeitung.

Die Möglichkeit des Multiprocessings stellt neue Herausforderungen an Computersystementwickler. Während in einer Mainframe-Umgebung alle Teile des Programms für eine einzige Umgebung entwickelt wurden, können die Entwickler bei Multiprocessingsystemen die jeweilige Umgebung wählen, in denen spezielle Aspekte eines Programms laufen werden.

Obwohl diese Freiheit bei der Verteilung des Programms in hocheffektive Anwendungen resultiert, müssen die Programmierer jetzt Programme konstruieren, die zur Ausführung in vielen Hardwareumgebungen ausgelegt sind.

Die Aufgabe des Programmierens von Multiprocessingsystemen schließt schwierige Probleme des Austauschens von Daten zwischen verschiedenen und inkompatiblen Umgebungen ein. Zum Beispiel ist eine Datei, die in einem PL/I-Format gespeicherte Daten enthält, nicht von einem Programm lesbar, daß in einer anderen Sprache, wie beispielsweise C oder COBOL geschrieben ist. Ein Programmierer muß spezielle Software entwickeln, um die Kommunikationsprobleme zu handhaben, die Multiprocessingsystemen inhärent sind.

Zusätzlich muß ein Programmierer die verschiedenen verteilten Programme in der Sprache einprogrammieren, die von der jeweiligen Umgebung unterstützt wird. Zum Beispiel müssen, falls die PC-Prozessoren nur Programme unterstützen, die in der Sprache C geschrieben sind, jene Teile des Programms in C sein. Gleichzeitig kann es sein, daß die anderen Teile des Programms, wie beispielsweise die auf dem Mainframe, in einer anderen Programmiersprache geschrieben sein müssen.

Das Programmieren in einer Multiprozessorumgebung ist sehr komplex und zeitaufwendig. Die Personalanforderungen zum Entwickeln einer Anwendung in einer Multiprozessorumgebung allein können die Aufgabe wegen der Kostenseite unerschwinglich machen. CASE-Einrichtungen wurden entwickelt, um einige der Belastungen zu verringern, die die Multiprocessingarchitektur dem Programmierer auferlegt hat.

Traditionell ermöglichen CASE-Einrichtungen einem Benutzer, ein logisches Konzept für ein Programm in einer höheren Sprache einzugeben, das dann in einen Code in einer jeweiligen Computersprache übersetzt wird. Die derzeit verfügbaren CASE-Einrichtungen geben dem Systementwickler jedoch kein komplettes einheitliches System für die Planung, die Umsetzung und die

Pflege einer Softwareanwendung an die Hand. Die derzeitigen Einrichtungen unterstützen nicht die Entwicklung eines Programms an einem zentralen Ort, wobei ein übersetzter Code erzeugt und an verschiedene Umgebungen verteilt wird. Die derzeitigen CASE-Tools stellen auch kein Verfahren zur Wiederverwendung von Teilen von zuvor entwickelten Anwendungen bereit, die in der in der Entwicklung befindlichen Anwendung verwendbar sein können. Die derzeitigen CASE-Einrichtungen stellen keine geeigneten Test- und Entstör-Tools bereit.

Eines der Dilemmas beim Entwickeln eines CASE-Tools ist, daß sich die Ausgabe, die dem Benutzer für die Planungsphase des Projekts zur Verfügung gestellt werden muß, sich von der Ausgabe unterscheidet, die später während der Umsetzung benötigt wird. Frühe CASE-Tools neigten dazu, die Unterstützung einer dieser beiden Kategorien von Aktivitäten zu betonen und die andere zu vernachlässigen.

Bei den früheren Planungs-orientierten CASE-Tools war die Ausgabe einer Entwicklerarbeit mit dem Tool ein Bild. Womit ein Entwickler endete, war Dokumentation, die verwendet werden konnte, um tatsächlich ein System zu erzeugen, aber nicht das tatsächliche System selbst. Zusätzlich mußte der Entwickler, falls während der Umsetzungsphase ein Anwender einen Grund aufdeckte, um irgendeinen Aspekt des Programms zu ändern, zurückgehen und manuell die Änderung identifizieren und die Dokumentation aktualisieren -- die Dokumentation war nutzlos, solange sie nicht ständig aktualisiert wurde. Die tatsächliche Umsetzung des Systems sowie jegliche Änderungen wurden unter Verwendung traditioneller Verfahren ausgeführt. Der einzige Vorteil bei der Verwendung eines Planungs-orientierten CASE-Tools war, daß es annehmliche vollständige Dokumentation erzeugte, die bei der Planung in den frühen Stadien wirklich hilfreich war.

Die zweite Familie der CASE-Tools war umsetzungsorientiert. Diese CASE-Tools erlaubten es einem Entwickler, ein Schaubild

eines vollständigen Modells zu zeichnen, aber nicht eines Stück eines Moduls. Im wesentlichen konnte der Entwickler ein Bild des Modells erhalten, an dem der Entwickler gerade arbeitete. Wenn der Entwickler das Modell in das Mainframe-Endlager entlud, ersetzte das Modell das Modell, das dort jeweils bereits gespeichert war. Die tatsächliche Dokumentation dieser Systeme war primitiv und unflexibel.

In einem Artikel überschrieben "An Approach to the Support of Software Evolution" von I. Sommerville und R. Thompson, Computer Journal, Vol. 32, 0.5, Oktober 1989 ist ein System zur Unterstützung aller Stadien des Softwarelebenszyklus beschrieben. Insbesondere unterstützt das System die Kommunikation und Dokumentation der Struktur des Softwaresystemprojekts, während es sich entwickelt. Das System erzeugt jedoch keine tatsächlichen Computerprogramme.

Zusammenfassung der Erfindung

Die vorliegende Erfindung ist eine CASE-Einrichtung, die ein Verfahren und eine Vorrichtung zum Planen, Umsetzen und Pflegen von Software bereitstellt, welche in mehrteiligen Hardwareumgebungen läuft -- in einer Verteilung, die für den Endbenutzer transparent ist.

Ein Aspekt dieser Erfindung ist ein Verfahren zur Modellbildung der Datenstruktur und der Daten, die von dem Programm verwendet werden, durch eine Elemente-Beziehungen-Modelliertechnik.

Ein anderer Aspekt ist eine relationale Datenbank, in der Daten gemäß dem Elemente-Beziehungen-Modell gespeichert werden.

Ein anderer Aspekt ist die höhere Regelprache, in der die Logik eines Programms in hochmodularer Form festgelegt werden kann und die die Einfachheit der Wiederverwertbarkeit fördert.

Die Erfindung stellt die Möglichkeit zur Erzeugung von Quellcode

für das Programm in einer Sprache bereit, die von den verschiedenen Hardwareelementen in dem Multiprocessingsystem unterstützt wird, unter Verwendung des Elemente-Beziehungen-Modells und der Module der höheren Regelsprache.

Um den Quellcode des Programms zu erzeugen, stellt die Erfindung Grundprogrammelemente zur Verfügung, um das Programm in einer Multiprocessingumgebung auszuführen. Diese Programmkomponenten und die niederen Routinen werden mit den Modulen der Regelsprache und den Datentypelementen kombiniert, um das Programm zu erzeugen.

Die Erfindung stellt weiterhin Einrichtungen bereit zum Verteilen, Zusammensetzen und Kompilieren des Programms, das unter Verwendung der CASE-Einrichtung der vorliegenden Erfindung entwickelt wurde, sowie zum Testen und Modifizieren des Programms.

Kurzbeschreibung der Zeichnungen

Fig. 1 ist eine Darstellung eines dreifach unterteilten Computersystems.

Fig. 2 ist eine zeichnerische Wiedergabe eines Programmelements gemäß der vorliegenden Erfindung.

Fig. 2a ist eine zeichnerische Wiedergabe eines beispielhaften Entwicklungswerkbankmoduls, das in einer PC-Hardwareumgebung angeordnet ist.

Fig. 3 ist eine zeichnerische Wiedergabe einer beispielhaften Elementbeziehung gemäß der vorliegenden Erfindung.

Fig. 4 ist eine zeichnerische Wiedergabe einer beispielhaften Elementzerlegung gemäß der vorliegenden Erfindung.

Fig. 5+6 sind zeichnerische Wiedergaben des Elemente-Beziehungen-Modells für Anwendungsprogramme gemäß der

vorliegenden Erfindung.

Fig. 7 ist eine zeichnerische Wiedergabe des Prozeßflusses für ein beispielhaftes Programm unter Verwendung der Elemente-Beziehungen-Modelliertechniken gemäß der vorliegenden Erfindung.

Detaillierte Beschreibung der Erfindung

A. Hardware

Eine typische Hardwarekonfiguration für ein Computersystem, das die CASE-Einrichtung der vorliegenden Erfindung verwendet, ist in Fig. 1 gezeigt. Die Figur beschreibt ein "dreifach unterteiltes" Computersystem, das nach den drei unterschiedlichen Typen der vernetzten Prozessoren benannt ist: dem Mainframe-Computer 1, z. B. einem IBM 3090, einem Mini- oder Supermini-Computer 2, z. B. einem IBM S/88 oder Stratus, und einer Mehrzahl von Mikro-Computerarbeitsplätzen 3, z. B. IBM PC Arbeitsplätzen. Ein System, das die CASE-Einrichtung der vorliegenden Erfindung verwendet, ist nicht nur auf diese Hardwarekonfiguration beschränkt. Zum Beispiel kann eine ähnliche CASE-Einrichtung aufgebaut werden, um einen Code zu entwickeln, der in einer zweifach unterteilten Umgebung verteilt wird, wie beispielsweise in einem System, das nur einen Mainframe und PC-Arbeitsplätzen verwendet, oder in einer zweifach unterteilten Umgebung, die einen Mainframe-Computer und einen Minicomputer aufweist, wobei diese beide von nicht-intelligenten Endgeräten zugänglich sind. Die CASE-Einrichtung kann zugeschnitten werden, um zu jeder Hardwarekonfiguration zu passen. Die Grundprinzipien der Codeerzeugung, der Pflege der zentralisierten Datenbank und der Verwendung der auf Regeln basierenden Sprache, die in Verbindung mit anderen Entwicklungstools verwendet wird, welche von dieser Erfindung aufgezeigt werden, sind dieselben, unabhängig von einer speziellen Hardwarekonfiguration.

Die Flexibilität bezüglich der Hardwarekonfiguration ist ein

Vorteil der CASE-Einrichtung gemäß der vorliegenden Erfindung gegenüber solchen CASE-Einrichtungen, die derzeit verfügbar sind. Die CASE-Einrichtung der vorliegenden Erfindung ermöglicht flexible architektonische Konzepte durch Erzeugung einer zentralisierten Datenbank, genannt Endlager, wo das Konzept und die Logik des Programmoduls gespeichert wird und von wo der übersetzte Code automatisch an die verschiedenen Hardwareumgebungen verteilt wird.

Typische Prozessoren, die für jede der Stufen bei der beispielhaften dreifach unterteilten Hardwarekonfiguration bevorzugt sind, sind der IBM-Mainframe, der unter der Handelsmarke "Modell 3090" verkauft wird und auf dem das IBM NVS/XA-Betriebssystem läuft; der Stratus Mini-Computer, der unter der Handelsmarke "Modell XA 2000" verkauft wird und auf dem das Stratus Computer, Inc. VOS Betriebssystem läuft; und der IBM Mikro-Computer, der unter der Handelsmarke "PS2" verkauft wird und der das Microsoft Corporation Betriebssystem ausführt, das unter der Handelsmarke "MS-DOS" verkauft wird. (Für weitere Informationen zu diesen Prozessoren und ihren Betriebssystemen wird der Leser auf die folgenden Publikationen verwiesen, die hiermit ausdrücklich durch Bezugnahme inkorporiert sind: "IBM System/370 Bibliography", Dokument Nr. 6024448; und "Introduction to VOS" der Stratus Computer, Inc., Dokument Nr. R 0001.)

B. <u>Elemente der CASE-Einrichtung</u>

Die CASE-Einrichtung der vorliegenden Erfindung weist eine Anzahl von programmierten Elementen auf, wie sie in Fig. 2 dargestellt sind: ein Endlager 4; eine Regelsprache 6; einen Kommunikationsmanager 8; eine Test-/Entstöreinrichtung 10; eine PC-Benutzer-Schnittstelleneinrichtung 12, einschließlich eines Regelzeichners 30; eine Arbeitsplatzversorgung, die einen Arbeitsplatzmanager 14, einen Fensterzeichner 16 und Arbeitsplatzumwandlung 18 aufweist; eine Dokumentationseinrichtung 20; und einen Systemeerzeuger 22 einschließlich eines Codeerzeugers 24, eines Datenzugangserzeugers 26 und einem Systemzusammenbauers 28. Obwohl die von der CASE-Einrichtung erzeugten

Programme zentral gespeichert werden, können die Programmelemente des CASE-Systems in jeder der Hardwareumgebungen angeordnet sein, die bei einer Konfiguration verwendet werden.

Das Endlager 4 ist eine zentrale Datenbank, die verwendet wird, um alle Informationen über sämtliche Anwendungsprogramme zu speichern, die mit der CASE-Einrichtung der vorliegenden Erfindung erzeugt werden. Das Endlager 4 kann in irgendeiner Hardwareumgebung existieren, die eine übliche relationale Datenbank unterstützt. Zum Beispiel kann das Endlager 4 in der dreifach unterteilten Umgebung, die in Fig. 1 gezeigt ist, auf dem Mainframe Computer 1 existieren, unter Verwendung des IBM-Systems DB2 für das Verwalten relationaler Datenbanken.

Zum Zweck einer bevorzugten Ausführungsform dieser Erfindung ist es bevorzugt, daß das zentrale Endlager in der Mainframe-Umgebung angeordnet ist und daß die relationale IBM-Datenbank, die unter der Handelsmarke "DB2" verkauft wird, verwendet wird. (Für Hintergrundinformationen zu DB2 wird der Leser auf die folgenden IBM-Publikationen verwiesen, die hiermit ausdrücklich durch Bezugnahme inkorporiert werden: "IBM DATABASE 2 Introduction to SQL" (Dokument Nr. GC26-4082); "IBM DATABASE 2 Reference" (Dokument Nr. SC26-4078); "OS-VS2 TSO, "Command Language Reference" (Dokument Nr. GC28-0646); "TSO Extensions Command Language Reference" (Dokument Nr. SC28-1307); "Interactive System Productivity Facility/Program Development Facility for MBS: Programs Reference" (Dokument Nr. SC34-2089); "Interactive System Productivity Facility/Program Development Facility of MVS: Dialogue Management Services" (Dokument Nr. SC34-2137) und "DB2 Application Programming Guide for TSO and Batch Users".)

Die in dem Endlager 4 gespeicherten Informationen schließen Modelle ein, die von der vorliegenden Erfindung bereitgestellt werden, um die Grundlage für ein Anwendungsprogramm und logische Module höheren Niveaus zu bilden, die von der vorliegenden Erfindung für die Verwendung bei der Erzeugung eines ausführ-

baren Programms definiert werden, wie deutlich werden wird. Für jedes Programm werden ein Datenmodell und ein Verarbeitungsmodell durch Verwendung eines Elemente-Beziehungen-Modelliersystems entwickelt und in dem Endlager 4 gespeichert. Die logischen Module höheren Niveaus, die in dem Endlager 4 gespeichert werden, sind in einer Regelsprache geschrieben, die von der vorliegenden Erfindung wie unten beschrieben definiert wird. Die Informationen sind umgebungsabhängig und strukturiert, um einen hohes Maß an Wiederverwendbarkeit bereitzustellen.

Programmierer geben Informationen unter Verwendung einer üblichen Datenbanksprache, die von der Hardware unterstützt wird, in das Endlager 4 ein. Zum Beispiel, wenn das Endlager 4 aus einer IBM DB2-Datenbank aufgebaut ist, würde ein Programmierer die DB2 Structured Query Language, SQL, verwenden, um ein Modell der Anwendung zu bilden. In einem anderen unten beschriebenen Beispiel könnten grafische Schnittstellenmodule für den Benutzer vorgesehen sein, um die Informationen einzugeben.

Die Regelsprache 6 ist eine höhere Sprache, die es dem Benutzer ermöglicht, die Logik eines Programms festzulegen, unabhängig von den Hardwarevorrichtungen, die von dem System verwendet werden. Die in der Regelsprache 6 geschriebenen Programmodule werden von dem Codeerzeuger 24 in einen Computercode übersetzt, der für die Ausführung in einer Umgebung geeignet ist, in der die Module laufen sollen. Die Regelsprache wird unten vollständiger beschrieben werden.

Der Kommunikationsmanager 8 führt den Laufzeit-Übertrag von Informationen zwischen den Hardwareumgebungen durch. Zum Beispiel würde der Kommunikationsmanager 8 Router und Protokolle verwenden, um den Datenübertrag zwischen einem Mainframe, einem Minicomputer und PC-Arbeitsplätzen zu handhaben.

Die Testeinrichtung 10, die ein Regelbetrachtungsmodul aufweist, ermöglicht es den Programmierern, den Programmcode durchzugehen und zu entstören. Das Regelbetrachtungsmodul kann Testdaten

erzeugen und eine Regressions- und Beanspruchungstestung einer Anwendung ermöglichen. Das Regelbetrachtungsmodul wird unten vollständiger beschrieben.

Das PC-Front-End 12 ermöglicht es einer PC-gestützten Grafikschnittstelle, für alle CASE-Tool-Funktionen verwendet zu werden. Der Regelzeichner 30 ermöglicht es dem Programmierer, Programmodule durch Manipulieren grafischer Darstellungen der Regelsprachenanweisungen zu konstruieren. Das PC-Front-End 13 beseitigt durch Anbieten aller PC-Funktionen in Menüs für die Notwendigkeit, daß der Programmierer, eine Betriebssystemsprache, beispielsweise DOS, kennt.

Der Arbeitsplatzmanager 14 hilft beim Managen der Benutzerschnittstelle in einer PC-Umgebung. Das Fensterzeichnermodul 16 ist ein Tool, das dem Entwickler hilft, Benutzerschnittstellenbildschirme für Anwendungen zu erzeugen. Die Arbeitsplatzumwandlung 9 managt die Anzeige und Validierung von Bildschirminformationen.

Das Arbeitsplatzmanagermodul 14 arbeitet in Kombination mit kommerzialisierten Programmen, um eine Benutzerschnittstelle vorzusehen, so wie beispielsweise Microsoft Windows. Alle Komplexitäten der Verwendung eines kommerziellen Enwurfstools, wie Microsoft Windows, werden von dem Arbeitsplatzmanager 14 gemanagt.

Beim Planen einer Anwendung oder Teilen davon zur Ausführung auf einem "IBM PC" ist es bevorzugt, daß das Microsoft Cooperation Betriebssystem, das unter der Handelsmarke "Microsoft Windows" verkauft wird, verwendet wird. (Für Hintergrundinformationen über Microsoft Windows wird der Leser auf die folgenden Microsoft Corporation-Veröffentlichungen verwiesen, welche hiermit durch Bezugnahme inkorporiert werden: "Microsoft Windows Programmer's Utility Guide"; "Microsoft Windows Application Style Guide"; "Microsoft Windows Programmer's Reference"; und "Microsoft Windows Quick Reference".)

Die Dokumentationseinrichtung 20 erzeugt die gesamte technische Dokumentation zu einem in Entwicklung befindlichen Programm. Die Dokumentation umfaßt funktionelle Zerlegungen des Systems und hierarchische Listungen der Regelsprachenmodule.

Der Systemerzeuger 22 umfaßt den Codeerzeuger 12, den Systemzusammensetzer 28 und den Datenzugangserzeuger 26. Der Codeerzeuger 24 übersetzt die Regelsprachenmodule in einen Code in einer geeigneten Programmiersprache. Der Systemzusammensetzer 28 bringt die verschiedenen einprogrammierten Programmelemente zusammen, um ein Anwendungsprogramm auszubilden. Der Datenzugangserzeuger 26 ermöglicht es dem Programm, auf Daten über die Hardwareumgebungen hinweg zuzugreifen. Die Codeerzeugung und Programmausführung wird unten vollständiger diskutiert werden.

Das PC-Front-End 12, der Arbeitsplatzmanager 14 und der Kommunikationsmanager 18 bilden zusammen ein Set von Entwicklungswerkbankmodulen 34 aus, die es dem Benutzer ermöglichen, Daten- und Prozeßmodelle für eine Anwendung und Regelsprachenmodule zu planen und diese zu kombinieren, um ein voll funktionsfähiges Anwendungsprogramm zu kreieren.

Auf der Entwicklungsplattform bei der oben beschriebenen beispielhaften Konfiguration erfolgt der Zugriff auf alle Umgebungen über das PC-Front-End 12. Dort können die Entwickler unter Verwendung der Regelsprache ihre Anwendungen einprogrammieren, und die CASE-Einrichtung wird wie erforderlich automatisch ursprünglichen Code für jede Umgebung erzeugen.

Für die letzten Phasen der Entwicklung ist es jedoch zum Kompilieren und Testen der Anwendungen erforderlich, jeweils zu einem Front-End-Modul für die geeignete Ausführungsumgebung zu schalten. Bezüglich dieser Anwendungsentwicklungsaktivitäten wird auf jede Umgebung über ein umgebungsspezifisches Front-End zugegriffen. Die umgebungsspezifische Aktivität wird unten vollständiger beschrieben werden.

Das Softwareverteilsystem 32 automatisiert und steuert die Verschiebung einer Anwendung. Das System managt die Freigabe von Software an Zielcomputer. Das Softwareverteilsystem 32 löst das Problem der Synchronisierung der Verteilung von Software, die z. B. auf Hunderten von Personal Computern angeordnet ist. Für eine detailliertere Beschreibung eines Softwareverteilsystems siehe die Internationale Anmeldung Nr. PCT/US90/_____, überschrieben "Software Distribution System" und am selben Tag wie diese Anmeldung eingereicht im Namen von Norman Chin et al., die hiermit ausdrücklich durch Bezugnahme inkorporiert wird.

C. Elemente-Beziehungen-Modellbildung

1.) Elemente und Beziehungen

Ein Merkmal der CASE-Einrichtung der vorliegenden Erfindung ist das Systememodell. Zum Planen einer Anwendung unter Verwendung der CASE-Einrichtung muß ein Entwickler die Anwendung in spezifische logische Teile zerlegen und diese zu einem Programm zusammensetzen. Die Daten, die von einem Programm verwendet werden, sind in dem Endlager 4, Fig. 2, gemäß einem Elemente-Beziehungen-Modell der vorliegenden Erfindung gespeichert. Die verschiedenen Teile einer Anwendung werden als Elemente ausgedrückt und durch Beziehungen verknüpft.

Allgemein definiert ist ein Element irgendetwas Reales oder Abstraktes, über das Informationen aufgezeichnet sind. Die Informationen sind in einem Set von als Attribute bekannte Eigenschaften organisiert. Zum Beispiel könnten die über Beschäftigte einer Firma gesammelten Informationen in einem Element angeordnet sein, das "Beschäftigter" bezeichnet wird. Die Attribute für dieses Element könnten sein: Name, Sozialversicherungsnummer, Wohnadresse, Alter, Geburtstag, Abteilung usw.. Ein Element, das "Organisation" genannt wird, würde solche Attribute wie Organisationsname, Adresse, Art der Organisation (wie beispielsweise Partnerschaft oder Gesellschaft) usw. umfassen. Diese Daten werden in einer Datei in dem Endlager 4 (siehe Fig. 2) gespeichert.

Eine Verbindung zwischen Elementen ist als Beziehung bekannt. Zum Beispiel ist in Fig. 3 das Element Organisation 64 jetzt mit dem Element Beschäftigter 58 über die Beziehung "Beschäftigt" 66 verknüpft. Beziehungen werden ebenfalls durch Attribute definiert.

2.) Funktionelle Planung

Die funktionelle Planungsphase beginnt mit der Aufgabe der Modellbildung, wobei zwei Aufgaben auszuführen sind: Datenmodellbildung und Prozeßmodellbildung.

Die Datenmodellbildung ist das Verfahren zum Erzeugen eines Elemente-Beziehungen-Modells für die echten Daten, die von einem Anwendungsprogramm zu verwenden sind. Das Verfahren schließt die Identifizierung eines Elements, wie beispielsweise einer "Gesellschaft", und die Zerlegung des Elements in Unterelemente und Attribute ein. In Fig. 4 ist das Beispielselement, die Gesellschaft 70, in Unterelemente zerlegt: Produkt 72, Personal 74 und Kunde 76. "Kunde" besteht aus den Attributen Name 78, Adresse 80 und dem Element Auftrag 82. Das Element Auftrag 82 ist zusammengesetzt aus den Attributen: Nummer 84, Datum 86 und Status 88.

Nachdem der Programmierer einmal ein Modell der Elemente der echten Daten gebildet hat, verknüpft er oder sie die Elemente unter Verwendung der Beziehungen, wie beispielsweise der Beziehung Beschäftigt 3, die in Fig. 3 illustriert ist.

Im Modellformat werden die Daten in dem Endlager 4 (siehe Fig. 2) gespeichert. Unter Verwendung der Elemente-Beziehungen-Modelliertechnik können Daten für jegliche Anwendungen gespeichert und bei nachfolgenden Anwendungen schnell wiederverwendet werden.

Die Prozeßmodellbildung ist das Verfahren des Aufbaus eines Modells eines Anwendungsprogramms unter Verwendung eines Elemente-Beziehungen-Modells. Die CASE-Einrichtung der vorlie-

genden Erfindung erfordert, daß alle Anwendungen zuerst auf ein Elemente-Beziehungen-Modell reduziert werden, bevor eine Logik des Programms festgelegt wird. Das Elemente-Beziehungen-Modell ist nicht das tatsächliche Programm. Es ist von den Programmmodulen getrennt, die in der Regelsprache geschrieben sind. Die CASE-Einrichtung verwendet jedoch das Elemente-Beziehungen-Modell, um die verschiedenen Programmodule miteinander zu verknüpfen sowie um das Programm aufzubauen und über das Multiprozessorsystem zu verteilen. Elemente speichern Informationen über den Fluß eines Programms, über die verwendeten Datenstrukturen, über die Benutzerschnittstelle, über die von den Modulen verwendeten Umgebungen und über die Multiprocessinganforderungen des Programms.

Zusätzlich stellt das Elemente-Beziehungen-Modell eine Wiedergabe der höheren Planung des Programms aus den Elementen bereit. Die CASE-Einrichtung kann eine technische Dokumentation aus dem Prozeßmodell herstellen. Für eine gegebene Anwendung gibt das Elemente-Beziehungen-Modell implizit eine Zusammenfassung einer Karte der höheren Struktur und einer detaillierte Beschreibung der Logik und der Datenanforderungen wieder, die notwendig sind, um das Programm laufen zu lassen.

Das prozeßmodellierende Verfahren erlaubt es Benutzern der CASE-Einrichtung, die Effizienz durch Wiederverwendung von Programmelementen zu maximieren. Das aus dem Elemente-Beziehungen-Modell resultierende Programm ist hoch modular. Weil alle Programme, die mit der von dieser Erfindung bereitgestellten CASE-Einrichtung entwickelt wurden, dieselben Elementtypen und Beziehungen verwenden, ist es wahrscheinlich, daß viele der Programmelemente wiederverwendet werden können.

D. Anwendungsmodellelemente und -beziehungen

Als Beispiel der vorliegenden Erfindung werden Programme, die unter Verwendung der CASE-Einrichtung aufgebaut wurden, in 10 Elementtypen und 11 Beziehungen zerlegt, wie in den Fig. 5 und 6 dargestellt ist. Für jedes Element und jede Beziehung, die

eine Anwendung definieren, wird eine Liste von Attributen und Informationen geführt.

1.) Elemente

Das Element Funktion 90, Fig. 5, ist eine Auflistung aller Anwendungsprogramme, die sich aktuell auf dem System befinden. Eine Funktion 90 ist definiert durch die folgenden Attribute: Funktionsname, Testbeschreibung und Anwendungsidentifikation.

Ein Prozeß 92, Fig. 5, ist eine logische Unterteilung einer Funktion 90. Eine Funktion 90 wird typischerweise in zwei oder mehr Prozesse 92 zerlegt. Die Prozesse 92 können in Prozesse 92 niedrigeren Niveaus (Unterprozesse) zerlegt werden. Wenn ein Prozeß 92 auf einen anderen Prozeß 92 bezogen ist, ist die Beziehung immer hierarchisch. Es gibt drei Prozeßtypen: Wurzel, Blatt und Knoten. Solche Beziehungen definieren die Zerlegung eines Untersystems in andere. Das Element Prozeß 92 ist definiert durch die Attribute: Prozeßname, Beschreibung, Menübeschreibung, Unterprozeßmenütyp und Sequenznummer.

Beim Laufenlassen läuft ein Prozeß 92 entweder als Vordergrund- oder als Hintergrundprozeß. Ein Vordergrundprozeß ist ein Prozeß, der interaktive Kommunikation mit dem Endbenutzer ausführt. Zum Beispiel kann ein Vordergrundprozeß grafische Funktionen aufweisen, die in einer PC-Arbeitsplatzumgebung 3, Fig. 1, befindlich sind. Dies ist der typische On-line-/Echtzeitprozeß. Zusätzlich können die Vordergrundprozesse synchron mit Modulen in anderen Umgebungen kommunizieren, und sie können asynchron nicht abgerufene Daten empfangen.

Ein Hintergrundprozeß wird, nachdem er einmal gestartet ist, seine Eingabe verarbeiten, dann stoppen oder auf weitere Eingabe warten. Dies kann ein schubweiser Prozeß sein, der beispielsweise auf dem Mainframe läuft und der eine Eingabe aus einer Datei erhält, oder es kann ein kontinuierlicher Prozeß sein, der on-line läuft, beispielsweise weil er aus einer Warteschlange liest.

Regeln sind die Verfahrensbeschreibungen der Logik eines Prozesses. Diese Logik ist in einer die Regelsprache genannten höheren Sprache festgelegt. Die Regelsprache basiert auf den Prinzipien der strukturierten Planung und Programmierung. Die Syntax der Sprache stellt zusammen mit den Beschränkungen, die in die Architektur eingebaut sind, einen hochmodularen und präzisen Systembeschreibungsprozeß sicher. Die Regelsprache wird unten im Detail beschrieben.

Unter Verwendung der Regelspracheanweisungen erzeugt die CASE-Einrichtung einen Quellcode für die verschiedenen Umgebungen. Die CASE-Einrichtung der vorliegenden Erfindung erzeugt alle Codes, die notwendig sind, um Intersystemkommunikation (z. B. wenn eine Regel in einer Umgebung eine Regel in einer anderen aufruft), Interprozeßkommunikation (z. B. zwischen verschiedenen Prozessen oder Regionen auf derselben Maschine), Programm-zu-Programm-Verknüpfung und eine Benutzerschnittstelle zu betreiben.

Bei dem Elemente-Beziehungen-Modell werden nur Informationen über die Regel in einem Regelelement 94 gespeichert -- nicht die Regelspracheanweisungen. Das Regelelement 94, Fig. 5, Fig. 6, wird durch seine Attribute definiert: Regelname, Regelbeschreibung, Ausführungsumgebung und Art der Ausführung (beispielsweise synchron oder asynchron).

Wie bei den Prozessen gibt es drei Kategorien von Regeln: Wurzel-, Grenz- und Knotenregeln. Eine Wurzelregel wird von einem Prozeß 92 aufgerufen. Wurzelregeln haben keine Benutzer-Eingabe/Ausgabe-Möglichkeit. Eine Knotenregel ist jede Regel, die keine Wurzel- oder Grenzregel ist.

Grenzregeln liegen im Randbereich einer neuen Computerumgebung. Diese Regeln sind in eine spezielle Kategorie eingruppiert, weil der Kommunikationsmanager 8, Fig. 2, alle Grenzregeln ausführen muß und weil ihre Eingabe/Ausgabedaten zur Anpassung an Wechsel in der Umgebung konvertiert werden müssen.

Die Regel-Beziehungen werden unter Verwendung von Regelelementen 94 und den Verwendet-Beziehungen 108 modelliert.

Komponenten sind Module von Code, der in irgendeiner dem CASE-Entwicklungssystem bekannten Programmiersprache der dritten Generation geschrieben ist, wie beispielsweise in C, PL/I oder COBOL. Komponenten werden verwendet, um Funktionen auszuführen, die nicht von der Regelsprache gehandhabt werden, wie beispielsweise Berechnungen, Datenbankzugriff und Anrufe des Betriebssystems. Die CASE-Einrichtung nimmt eine "Black Box"-Struktur für die Komponenten an. Eine Black Box hat feste Eingaben und Ausgaben. Ist eine bestimmte Eingabe gegeben, erfolgt immer eine vorbestimmte Ausgabe. Dieselbe Analogie trifft auf die Komponenten zu. Die Komponenten haben explizit festgelegte Eingaben und Ausgaben.

Das Komponentenelement 96, Fig. 5, Fig. 6, enthält im Unterschied zu dem oben beschriebenen Komponentenmodul die folgenden Attribute: Komponentenname; Komponentenbeschreibung; Sprachenname (Programmiersprache), in der der Quellcode der Komponente geschrieben ist); und Ausführungsumgebung.

Fenster definieren die Benutzerschnittstelle. Sie legen fest, welche Daten von einem Benutzer zu akzeptieren sind, wie sie anzuzeigen sind und wie sie zu akzeptieren sind. Das Fensterelement 98 wird verwendet, um alle Informationen über die Benutzerschnittstelle zu speichern. Die Fensterinformationen werden von dem Codeerzeugungsmodul 24, Fig. 2, und dem Arbeitsplatzumwandlungsmodul 18, Fig. 2, verwendet.

Ein Fensterelement 98, Fig. 5, weist bei dem Elemente-Beziehungen-Modell die folgenden Attribute auf: Fenstername, Beschreibung. Der Fenstername enthält den Namen einer Maskendatei, die ebenfalls in dem Endlager gespeichert ist und die Informationen enthält, welche notwendig sind, um eine grafische Schnittstelle unter Verwendung eines grafischen Schnittstellenprogramms, wie Microsoft Windows, herzustellen.

Das View-Element 100, Fig. 5, Fig. 6, ist ein nützlicher Gruppiermechanismus für das Speichern von Datentypvariablen, die von der Regelsprache verwendet werden. Der View 100 ist ein hierarchischer Set von benamten Skalaren oder Aggregatwerten.

Die Views 100 werden innerhalb eines typischen Systems auf drei verschiedene Weisen benutzt. Ein Datei-View 100A gibt einen Speicher der Datenkonstrukte wieder, die in einer Datendatei gesichert sind. Die Datenkonstrukte, die verwendet werden, um die Eingabe und Ausgabe für die Regeln und Komponenten zu beschreiben, sind modulare Views 100. Die Datenstrukturen, die verwendet werden, um die Benutzerschnittstelle zu handhaben, werden als Fenster-Views 100 bezeichnet.

Feldelemente 102, Fig. 5, Fig. 6, sind die Grunddatenelemente, die die in einem Programm verwendeten Variablen aufweisen. Das Feldelement 102 speichert alle Informationen über Datenelemente, unabhängig von der Umgebung, in der das Datenelement verwendet wird. Die Attribute dieses Elements bestimmen das Format, Editierfestlegungen, Bericht- und Bildschirmüberschriften und andere generische Informationen über ein bestimmtes Datenelement.

Ein Set 116, Fig. 6, ist wie ein View 100, Fig. 5, Fig. 6, ein nützlicher Gruppiermechanismus, der verwendet wird, um zueinander in Beziehung stehende Literale oder Konstanten zu speichern. Wertelemente 118, Fig. 6, sind symbolische Wiedergaben von Literalen oder Konstanten. Wertelemente 118 nehmen durch symbolische oder englische Namen Bezug auf spezifische Datenwerte oder stellen die Möglichkeit hierzu bereit. Dies beseitigt die Notwendigkeit, bestimmte Literalwerte in den Regeln fest zu programmieren.

Die Daten werden in Dateien gespeichert. In einem Elemente-Beziehungen-Modell werden Datendateien betreffende Informationen in einem Dateielement 120, Fig. 6, gespeichert.

2.) Beziehungen

Die zehn ein Programm wiedergebenden Elemente sind durch eine von elf Beziehungen miteinander verknüpft.

Die Löst-auf-Beziehung 104, Fig. 5, beschreibt die Zerlegung von Funktionen 90 zu Prozessen 92. Ihre Attribute sind: Funktionsname (erster Teilnehmer); Prozeßname (zweiter Teilnehmer); und Sequenznummer (für eine Menüanzeige). Die Prozesse lösen sich ebenso in weitere Unterprozesse auf.

Die Definiert-Beziehung 106, Fig. 5, wird verwendet, um die Beziehung zwischen einem abstrakten Prozeß 92 und den Regelelementen 94 zu beschreiben. Ihre Attribute sind: Prozeßname und Regelname. Prozesse können auch von anderen Prozessen abhängen.

Die Verwendet-Beziehung 108, Fig. 5, wird verwendet, um die Verknüpfung zwischen einem ausführbaren Modul und einem anderen zu beschreiben (beispielsweise Regel verwendet Regel, Regel verwendet Komponente und Komponente verwendet Komponente). Die Attribute einer Verwendet-Beziehung sind: Modulname und Untermodulname.

Die Wandelt-um-Beziehung 110, Fig. 5, beschreibt die Verknüpfung zwischen einem Modul und einem Fensterelement. Die Attribute der Wandelt-um-Beziehung 110 sind: Regelname (erster Teilnehmer) und Fenstername (zweiter Teilnehmer). Die Funktion der Wandelt-um-Beziehung wird unten vollständiger beschrieben werden.

Die Besitzt-Beziehung 112, Fig. 5, verbindet Elemente mit den View-Elementen 100, die ihre logischen Datenschnittstellen beschreiben. Das Regelelement 94, das Komponentenelement 96, das Fensterelement 98 und das Dateielement 120, Fig. 6, können View-Elemente 100 besitzen. Die Attribute der Besitzt-Beziehung sind: Regel-/Komponente-/Fenster- bzw. Dateiname; Elementtyp (entweder Regel/Komponente/Fenster oder Datei); View-Name und View-Verwendung (entweder In, Out oder Inout für eine Benutzerschnittstellen-Eingabe/Ausgabe).

Die Schließt-ein-Beziehung 114, Fig. 5, verbindet Datengrößen miteinander, um Strukturen auszubilden. View-Elemente können Unter-Views 100 und Felder 102 umfassen. Die Attribute der Schließt-ein-Beziehung 114 sind: View-Name (erster Teilnehmer, View höherer Ordnung); View-/Feld-Name (zweiter Teilnehmer); Elementtyp (Elementtyp des zweiten Teilnehmers: entweder View oder Feld); Sequenznummer (verwendet, um Unter-Views und -Felder zu ordnen); Auftrittshäufigkeit (wird verwendet, um Arrays in Strukturen zu erzeugen.

Die Besitzt-Set-Beziehung 124, Fig. 6, verknüpft ausführbare Module mit den Literal-Set-Werten, auf die sie sich beziehen. Eine Regel kann einen Set durch eine lokale Erklärung unter Verwendung der Regelsprachen-DCL-Anweisung besitzen, wie unten diskutiert wird. Die Literalwerte werden dann automatisch in das Regelmodul eingeschlossen, wenn der Code generiert wird. Ein Komponentenelement 96 kann ebenfalls Sets besitzen. Die Attribute der Besitzt-Set-Beziehung 124 sind: Modulname; Modulelementtyp (beispielsweise Regel oder Komponente); Setname.

Die Enthält-Beziehung 126, Fig. 6, verknüpft literale Datengrößen mit einem gemeinsamen Setelement 116. Ein Setelement 116 enthält z. B. 18 Mitglieder-Werte in dem Wertelement 118. Die Attribute für die Enthält-Beziehung 126 sind: Setname; Wert; Symbolname; Sequenznummer.

Letztlich weist das CASE-System Dateibeziehungen auf: ein Dateielement 120, Fig. 6, wird eingegeben 132 von einem Feldelement 122; das Dateielement 120 liefert auch Informationen an eine andere Datei 120; und ein Datei-View-Element 105 beschreibt 128 ein Dateielement 120.

E. Prozeßmodellbildungsreduktionstechnik

Alle Anwendungen, die mit der CASE-Einrichtung entwickelt wurden, können auf einen Set von vorgewählten Elementen und Beziehungen reduziert werden, wie beispielsweise die oben beschriebenen Elemente und Beziehungen. Das Verfahren beginnt

mit dem Definieren der Funktionselemente 90, Fig. 5. Eine Funktion ist jedem Anwendungsprogramm zur Durchführung einer Aufgabe oder Funktion äquivalent. Das Funktionselement 1 enthält eine generalisierte Beschreibung der Anwendung. Eine Funktion wird in Prozesse zerlegt. Die Prozesse geben allgemeine Funktionen wieder, die das Programm ausführen muß.

Zum Beispiel könnte ein Prozeßelement 92 in einer Anwendung zur Durchführung eines Börsenhandels überschrieben sein "Terminhandel-Eingang". Ein Prozeßelement 92, Fig. 5, speichert Informationen, die den Prozeß beschreiben. Prozesse können in weitere Prozesse zerlegt werden. Bei jeder Zerlegung wird die Information über die neuen Prozesse in einem Prozeßelement 92 gespeichert.

Das Funktionselement 90 und die Prozeßelemente 92 erzeugen einen höheren Überblick über eine Anwendung. Während der Modellbildungsphase stellt die CASE-Einrichtung, die von dieser Erfindung vorgestellt wird, Tools zur Unterstützung der Entwicklung zur Verfügung. Der Programmierer kann von der CASE bereitgestellte Grafiken verwenden, um das Prozeßelement 92, das Funktionselement 90 und die Löst-auf-Beziehungen 104 festzulegen. Der Verwender sieht die Funktionen und Prozesse gelistet in Menüs. Die CASE-Einrichtung unterstützt ebenso den Analysierer und Verwender bei der Prozeßmodellbildung, indem sie erlaubt, daß diese Funktions-/Prozeßzerlegung beispielsweise von einem Analysierer und einem Programmierer in Form von Prototypen interaktiv ausprobiert werden kann. Der Analysierer kann seine Gedanken eingeben, Menüs erzeugen, die die Eingabe wiedergeben, und Eingaben (Änderungen) von dem Programmierer fordern, falls diese tatsächlich der Weg sind, auf dem sich der Benutzer vorstellt, daß ein Problem gelöst werden soll.

Nachdem einmal ein Anwendungsproblem auf eine Reihe von Funktionselementen 90 und Prozeßelementen 92 reduziert worden ist, beginnt die technische Planung. Bei der technischen Planung werden die Programmcodefestlegungen durch das Regelelement 94,

das Komponentenelement 96 und das Fensterelement 98 beschrieben.

Der wichtige Aspekt der technischen Planung ist die Festlegung der Regeln, Komponenten, Fenster und Views, die notwendig sind, um das Programm laufen zu lassen. Fig. 7 ist ein Beispiel eines Prozeßflußdiagramms, das Elemente-Beziehungen-Modellbildungs-Standards verwendet. Die eckigen Kästen 134, 136, 138, 140, 142, 144 in der Figur stehen für Prozeß- und Funktionselemente. Die Kapsel-förmigen Kästen 148-182 stehen für Regeln. Die ovalen Kästen 184, 186, 192, 194, 196 stehen für Komponenten. Die Kreise 188, 190 stehen für Fenster. Das Prozeßflußdiagramm wird verwendet, um die Information, die in den Regelelementen 94, den Komponentenelementen 96, den Fensterelementen 98 und den View-Elementen 100 (siehe Fig. 5) enthalten ist, sowie die Beziehungen zwischen ihnen festzulegen.

Bezugnehmend jetzt auf Fig. 2 sind dort Entwicklungswerkbankmodule 34 gezeigt, die zur Erzeugung des Elemente-Beziehungen-Modells und des Regelsprachencodes verwendet werden. Fig. 2A zeigt eine beispielhafte Umsetzung der Module 34 in einem dreifach unterteilten System, wobei die Entwicklungswerkbankmodule 34 in der PC-Umgebung 3 vorhanden sind. Die Entwicklungswerkbankmodule 34 weisen eine Reihe von grafikgestützten Tools auf, die einem Benutzer helfen, eine Anwendung zu planen und umzusetzen, indem sie einen vorgewählten Set von Elementen und Beziehungen und eine Regelprogrammiersprache bereitstellen. Alles, was ein Benutzer unter Verwendung der Entwicklungswerkbankmodule 34 plant, wird in einem lokalen Endlager 60 in der PC-Umgebung 3 gespeichert werden. Gegebenenfalls werden die Daten in dem lokalen Endlager 60 verwendet, um das zentrale Endlager 4 zu besetzen, das sich in der Mainframe-Umgebung 1 befindet. Alles, was Dokumentation oder Umsetzung eines Systems ausmacht, muß während des Entwicklungszyklus gegenüber dem zentralen Endlager 4 definiert sein. Letztlich wird die Anwendung von dem zentralen Endlager 4 beispielsweise an End-Benutzer-PC-Arbeitsplatzumgebungen (und andere Umgebungen falls erforderlich) verteilt.

Die Verwendung der Entwicklungswerkbankmodule 34 muß jedoch nicht zu jeder Zeit an das zentrale Endlager 4 angeschlossen sein. Aus Gründen der Ausführung ist es vorteilhaft zu vermeiden, bei dem zentralen Endlager 4 ständig nach Informationen anzufragen. Das Heraufladen von und Herunterladen zu dem zentralen Endlager erfolgt über den Kommunikationsmanager 8, der auf das lokale Endlager 60 zugreift.

Das lokale Endlager 60 ist eine Speichervorrichtung, die in der PC-Umgebung 3 angeordnet ist. Ein Datenbanktreibermodul 54 weist Software zum Speichern von Informationen und zum Zugreifen darauf auf. Ein Treiber 52 für das lokale Endlager ermöglicht dem Benutzer Zugriff auf die Entwicklungswerkbankmodule 34 und ermöglicht dem Entwicklungswerkbankmodul 34, auf das lokale Endlager 60 über das Datenbanktreibermodul 54 zuzugreifen.

Ein Hierarchie-Diagramm-Modul 38 ist in dem Entwicklungswerkbankmodul 34 vorgesehen, um es einem Benutzer zu ermöglichen, die Daten- und Prozeßmodule festzulegen, die das logische Fundament des Anwendungsprogramms bereitstellen. Das Hierarchie-Diagramm-Modul 38 könnte z. B. ein Elemente-Beziehungen-Diagramm, das zur Datenmodellbildung verwendet werden könnte -- beispielsweise zum Festlegen des Elemente-Beziehungen-Modells für die echten Daten, die von dem Anwendungsprogramm verwendet werden. Mit dem Elemente-Beziehungen-Diagramm kann ein Benutzer grafisch Elemente festlegen und die Elemente in Unterelemente und definierende Attribute zerlegen. Zum Beispiel wurde in Fig. 4 eine Gesellschaft 70 in Unterelemente Produkt 72, Personal 74 und Kunde 76 zerlegt. Das Elemente-Beziehungen-Diagramm stellt Module zum grafischen Festlegen solcher Elemente und Beziehungen bereit.

Die Entwicklungswerkbankmodule 34 könnten auch ein Matrixaufbaumodul 42 aufweisen, das es einem Benutzer ermöglicht, Abbildungen zwischen verschiedenen Arten von Elementen in Matrizenform zu speichern. Die Matrizenabbildungen werden typischerweise in der strategischen Planungsphase der Analyse verwendet, um die

Beziehungen zwischen dem Daten- und Prozeßfluß festzuhalten. Auch die erzeugte Matrix wird in dem lokalen Endlager 60 und dem zentralen Endlager 4 gesichert.

Das Fensterzeichnermodul 16 ermöglicht es einem Benutzer, Masken zu erzeugen, die die Schnittstelle sind, über die der Endbenutzer einer Anwendung kommuniziert. Die Masken des Fensterzeichnermoduls 16 bestimmen das "Aussehen" und das von einer Anwendung vermittelte "Gefühl".

Unter Verwendung des Fensterzeichnermoduls 16 kann der Verwender entweder eine neue Fensterdatei für das zugehörige Fensterelement erzeugen oder eine zuvor erzeugte Fensterdatei, die in dem Endlager 4 gespeichert war, modifizieren.

Unter Verwendung des Beispiels eines PC, der mit einem Microsoft Windows Betriebssystem arbeitet, stellt das Fensterzeichnermodul 16 die Software zur Verfügung, die für einen Benutzer notwendig ist, um in dieser Umgebung eine grafische Schnittstelle zu erzeugen.

Zum Beispiel könnte ein Benutzer auf das Fensterzeichnermodul 16 von der Entwicklungswerkbank aus zugreifen, wobei ein leeres Fenster erscheinen würde. Der Benutzer könnte dann eine Liste von Fensterdateien auswählen, die zuvor in dem lokalen Endlager 60 definiert wurden. Das Anklicken eines dieser Fensternamen bringt eine Tafel hervor, auf der man die Objekte "zeichnen" kann, die dem Endbenutzer der Benutzeranwendung präsentiert werden.

Zusätzlich zu dem Maskenschirm gibt es vornehmlich zwei Arten von Objekten, die zu manipulieren sind: Elementobjekte und lokale Objekte.

Elementobjekte sind Feldelemente 102 und View-Elemente 100, die definiert worden sind und sich in dem lokalen Endlager 60 befinden. Ein Views-Menü kann eine Liste der View-Elemente 100

bereitstellen, die an einem Fenster anhängen, das ein Benutzer zeichnet, gefolgt von den auswählbaren Feldelementen 102 und mehrfach auftretenden Unter-Views, die wiederum an diese angehängt sind. Ein Benutzer kann jedes dieser verfügbaren Objekte auswählen, sie auf der Maske anordnen und sie individualisieren.

Felder könnten als leere geklammerte Kästen erscheinen mit einer Bildschirmbezeichnung auf der linken Seite. Der Wert des Felds (falls vorhanden) erscheint zur Laufzeit zwischen den Klammern. Die Bildschirmbezeichnung wurde definiert, als das Element erzeugt wurde, und ist dem Feld in dem Endlager 4 zugeordnet. Ein Benutzer kann die Bildschirmbezeichnung als Textobjekt auf der Maske editieren. Falls das Bildschirmbezeichnungsfeld oder das Feldelement leer ist, wird stattdessen der Feldkurzname verwendet.

Diese Objekte können durch Anklicken der Objekte und Gedrückthalten der Maustaste, während das Objekt zu dem gewünschten Ort gezogen wird, über die Maske bewegt werden.

Lokale Objekte sind der Maske zugeordnet, aber nicht gegenüber dem lokalen Endlager 60 definiert. Der Benutzer kann lokale Objekte von einem Objektmenü auswählen.

Es gibt drei Arten von lokalen Objekten:
Steuerobjekte, die vom Endbenutzer mit einer Maus ausgewählt werden können. Steuerobjekte, die Druckknöpfe, Druckkästen, Menübalkengegenstände und Pull-down-Gegenstände aufweisen. Textobjekte, die auf der Maske als Beschriftungen oder Anweisungsnachrichten angeordnet werden können.

Durch Steuern der Anordnung der Bildschirmobjekte kann ein Benutzer ein Fenster aufbauen, das einem View-Element 100 zugeordnet ist, und diese Fensterfestlegung für Routineanwendungen in dem lokalen Endlager 60 und in dem zentralen Endlager 4 sichern. Die Fensterdateien, die bei diesem Beispiel erzeugt werden, werden benutzt, um die Benutzerschnittstelle

eines Anwendungsprogramms zu erzeugen. Die unten vollständiger beschriebene Umwandlungsfunktion stellt die Softwaremodule zum Verknüpfen der vom Benutzer festgelegten Fensterdatei mit den Fenstern bereit, die von den grafischen Schnittstellenroutinen bereitgestellt werden.

Das Regelzeichnermodul 30 stellt die Schnittstelle bereit, die es Benutzern erlaubt, Regelsprachenmodule zu erzeugen, die den Regelelementen 94 zugeordnet sind. Die Regelsprachenmodule und Regelelemente sind verknüpft. Jedes Regelelement enthält Informationen betreffend die Funktionalität der Regel und den Namen eines entsprechenden Regelsprachenmoduls. Die entsprechenden Regelsprachenmodule enthalten Anweisungen, um die Funktionalität des verknüpften Regelelements auszuführen. Typische Anweisungen, die Regelsprache aufweisen, werden unten genauer beschrieben.

Das Berichtzeichnermodul 40 wird verwendet, um Formate für Endbenutzerberichte zu entwerfen, in einer Weise, die derjenigen des Festlegens von Fenstern für grafische Schnittstellen unter Verwendung des Fensterzeichnermoduls 16 ähnlich ist.

Das Datenbankadministratormodul (DBA) 44 ist eine Kommunikationseinrichtung, die es einem Benutzer ermöglicht, Daten in dem lokalen Endlager 60 aufzufrischen und Änderungen in das zentrale Endlager 4 zu laden. Das DBA-Modul 44 wird auch verwendet, um Berichte über die Inhalte des lokalen Endlagers 60 zu erzeugen. Dieses Modul verwendet den Lokalendlagertreiber 52, den Datenbanktreiber 54 und das Kommunikationsmanagermodul 8 in der PC-Umgebung 3, um auf Daten in dem lokalen Endlager 60 zuzugreifen und um Daten zum und von dem zentralen Endlager 4 zu senden und zu empfangen. Unter Verwendung des DBA-Moduls 44 können Objekte und Attribute aus dem Endlager 4 zur Änderung oder Löschung ausgewählt werden. Das DBA-Modul 44 weist eine Systemsicherheitskomponente auf, die fordert, daß sich Benutzer bei der Mainframe-Umgebung 1 anmelden und daß nur bestimmte Klassen von Benutzern die Erlaubnis zum Löschen bestimmter

Objekte haben können. Die Auffrischungsmöglichkeit erlaubt Benutzern, Objekte aus dem Endlager 4 heraus zu laden.

Die Entwicklungswerkbankmodule 34 ermöglichen es den Benutzern, die Datenmodellierung festzulegen, die notwendig ist, um Elemente und Beziehungen festzulegen, sowie die Regel-, Komponenten-, Views-, Feld- und Fenstermasken festzulegen, die notwendig sind, um das letztendliche ausführbare Programm in den späteren Entwicklungsschritten zu erzeugen. Unter Verwendung der Entwicklungswerkbankmodule 34 können diese logischen Strukturen in dem Endlager 4 gespeichert werden.

F. Aufbauen des Programms

Bei fertigem Elemente-Beziehungen-Modell kann der Programmaufbau beginnen. Der Quellcode wird aus den Regelsprachenmodulen, den Komponenten, Fenstern, Dateien, Feldern, Views, Werten und Sets erzeugt, die in dem Endlager gespeichert sind. Die Regeln, Komponenten, Fenster und Dateien sind Programmodule, die sich von den Regelelementen 94, Komponentenelementen 96, Fensterelementen 98 und Dateielementen 120 unterscheiden, die in dem Elemente-Beziehungen-Modell (siehe Fig. 5 und 6) verknüpft sind. Diese Elemente speichern Informationen über das tatsächliche Programmodul, aus dem das Anwendungsprogramm erzeugt wird.

In dieser Stufe des Aufbaus wird die meiste Arbeit beim Aufbauen des Programms getan. Die Datenstrukturen werden durch die View-Elemente 100, die Feldelemente 102, die Setelemente 116 und die Werteelemente 118 definiert. Die Codeerzeugereinrichtung 24, Fig. 2, kopiert diese Elemente in diejenigen Programmodule, die mit ihnen über die Regelelemente 94 und Komponentenelemente 96 (siehe Fig. 5) in Beziehung stehen. Komponentenmodule müssen nicht von dem Programmierer aufgebaut werden, da sie zuvor programmierte Unterroutinen sind, die von der CASE-Einrichtung bereitgestellt werden, um grundlegende mathematische Operationen und Betriebssystemzugriffe auszuführen. Die CASE-Einrichtung erlaubt es jedoch, daß neue Komponenten aufgebaut und in dem Endlager 4, Fig. 2, gespeichert werden. Der Prozeß des Code-

erzeugens wird unten vollständiger beschrieben.

1. Wiederverwendbarkeitsanalyse

Noch vor der Erzeugung der Regelsprachenmodule, beispielsweise unter Verwendung des Regelzeichnermoduls, beschleunigt die CASE-Einrichtung die Programmentwicklung, weil sie ein Verfahren zur Wiederverwendung zuvor erzeugten Codes bereitstellt. Bei der Elemente-Beziehungen-Modellbildung kann viel derselben Logik einer Anwendung bei anderen Anwendungen wiederverwendet werden. Es ist einfacher, Code wiederzuverwenden, der bereits entwickelt und getestet worden ist, als ihn wieder zu erzeugen. Dies bedenkend, sollte eine Wiederverwendungsanalyse durchgeführt werden.

Bei dem in dem Endlager 4, Fig. 2, gespeicherten Elemente-Beziehungen-Modell können die Elemente zur Verwendung durch andere Elemente abgefragt werden. Zum Beispiel gibt bei einem Endlager 4, das in einer von IBM DB2 unterstützten Datenbank existiert, eine Where Used-(Wo benutzt)-Anfrage bezüglich des Feldelements 122, Fig. 5, eine Liste aller View-Elemente 100A aus, die das Feldelement 122 verwenden. Die Verwendung der View-Elemente kann weiter abgefragt werden um festzustellen, welche Regeln, Komponenten oder Fenster jene View-Elemente verwenden. Dieses Verfahren erlaubt es den Programmierern, zuvor einprogrammierte Regeln und Komponenten zu erkunden. Eine Find Where-(Finde wo)-Abfrage wird alle Elementattribute über die gesamte Datenbank nach bestimmten Worten oder Zeichenfolgen durchsuchen.

Zusätzlich ermöglicht es die CASE-Einrichtung Programmierern, ein Schlüsselwort für alle erzeugten Regeln, Komponenten und Felder zu definieren. Ein Suchbefehl kann ausgeführt werden, um Elemente zu lokalisieren, die ein bestimmtes Schlüsselwort besitzen. Ein Programmierer kann kompliziertere Suchen durchführen, wie eine solche, um alle Schlüsselwörter zu finden, die beispielsweise mit "CU" anfangen und mit "Pp" enden.

Nachdem ein Programmierer ein Modul lokalisiert hat, das wieder-

verwendbar sein kann, kann er oder sie weitere Informationen durch Durchsehen der Beschreibung oder anderer Attribute erhalten, die dem Element zugeordnet sind.

2. Die Regelsprache

Die Regelsprache ist eine höhere Programmiersprache, die jeden Standardfluß von Steuerkonstrukten unterstützt. Was an der Regelsprache ungewöhnlich ist, ist daß sie kein Mittel zur Beschreibung ausgearbeiteter Datenstrukturen erfordert und daß sie Datenkonstrukte aufweist, die auf Informationen in den Elemente-Beziehungen-Modulen zugreifen. Die Beschreibung aller Datenstrukturen, die von einem Programm innerhalb der CASE-Einrichtung verwendet werden, ist in den View- und Set-Elementen in dem Elemente-Beziehungen-Modell des Programms gespeichert. Alle Regeln, die von einem Programm verwendet werden, teilen sich Datenstrukturen. Das Sich-Teilen von Datenstrukturen sorgt für eine strikte Koordination zwischen den Daten, die von einem Regelsprachenmodul auf ein anderes übertragen werden. Diese Technik verhindert eine Hauptquelle von Programmfehlern -- eine Fehlpassung von Daten, die zwischen Unterroutinen übertragen werden.

Eine Regel besteht aus keiner oder mehr erklärenden Anweisungen gefolgt von keiner oder mehr ausführbaren Anweisungen.

Ein erklärter Datentyp muß einer der folgenden Typen sein: Smallint, Integer, Char, Varchar, Decimal, Signed Picture oder Like (ein bereits erklärter Gegenstand). Die erklärende Anweisung hat die Form:

```
DCL
     Erklärung; [Erklärung; ...]
ENDDCL
```

wobei die Erklärung ist:

```
Bezeichnung [ (s) ] [, Bezeichnung [ (s) ], ...]
erkläre Typ oder EXTERNe Bezeichnung [, Bezeichnung, ...]
SET
```

Die Datentypen werden vollständiger im Anhang C erklärt, der hieran angehängt ist.

Die Regelsprache unterstützt drei Arten von ausführbaren Anweisungen: 1) Zuordnungsanweisungen; 2) externe Steuerfluß-anweisungen; und 3) interne Steuerflußanweisungen:

Zuordnungs-anweisungen	externe Steuer-flußanweisungen	interne Steuer-flußanweisungen
MAP	USE	IF
OVERLAY	RETURN	CASEOF
CLEAR	CONVERSE	DO
	ASYNC	

Zuordnungsanweisungen ändern die Daten, die in Programmvariablen enthalten sind und von diesen gehalten werden. Die MAP Anweisung ordnet einen bestimmten Wert an einem Variablenpunkt an. Die OVERLAY-Anweisung ersetzt die Inhalte einer Variablen durch einen bestimmten Datengröße. Die CLEAR-Anweisung ersetzt ein numerisches Feld durch Null-Werten und ein Zeichenfeld durch Leer-Zeichen. Die Syntax der Zuordnungsanweisungen ist:

```
MAP Ausdruck TO Variable
OVERLAY Datengröße TO Variable
CLEAR Variable
```

Vier Anweisungen in der Regelsprache handhaben den externen Steuerfluß zwischen Programmodulen. Die USE-Anweisung ermöglicht es einem Programmodul, eine andere Regel oder Komponente aufzurufen. Sie ist einer Unterroutinen-Call-Anweisung in anderen Programmiersprachen ähnlich. Die Syntax ist:

```
USE MODULE Komponente
USE RULE Regel [NEST]
```

Die Next-Option zeigt an, daß alle Fenster, die die Regel aufrufen, in einen "Auftauch"-Modus gelangen, das heißt, daß sie über dem zuvor angezeigten Bildschirm angezeigt werden.

Die RETURN-Anweisung führt die Steuerung zurück zu der Regel, die die USE-Anweisung ausgeführt hat:

```
RETURN
```

Die CONVERSE-Anweisung stellt eine Kommunikation zwischen der Regel auf dem PC und der Benutzerschnittstelle bereit. Eine typische CONVERSE-Anweisung ist:

CONVERSE WINDOW Fenster

Die CONVERSE-Anweisung ruft zur Laufzeit verschiedene Module auf, die von der vorliegenden Erfindung bereitgestellt werden, um physikalisch die Umwandlungsbeziehung zwischen Fenster und Regel auszuführen. Bei dem Beispiel einer auf dem PC ansässigen Regel, die auf ein Fenster auf dem PC zugreift, werden die Umwandlungsmodule in der PC-Umgebung 3 ansässig sein und an das Betriebssystem dieser Umgebung ankuppeln, um eine grafische Benutzerschnittstelle bereitzustellen.

Die Funktion der Umwandlungsmodule ist es, die gesamten Bildschirmeingaben und -ausgaben für eine Regel zu managen. Die Module senden die Eingaben des Benutzers durch Belegung eines bestimmten Felds zu der Regel. Vor dem Übertragen der Eingaben des Endbenutzers zu der Regel, die das Fenster umwandelt, führt das Umwandlungsmodul eine Editierüberprüfung und Validierung jedes Eingabefelds durch, das durch die in dem Endlager definierten Attribute festgelegt ist. Falls z. B. ein bestimmter Set von Werten vorlag, der für ein gegebenes Feld zu einer Regel in Beziehung stand (z. B. falls ein Farbfeld nur gleich "rot" oder "blau" sein könnte, wäre jede Antwort "grün" ein Fehler).

Beim Erzeugen des Benutzerbildschirms ist es auch eine Funktion des Umwandlungsmoduls, an das Betriebssystem anzukoppeln, um die Benutzerschnittstellenbildschirme, die durch die Fensterdateien festgelegt sind, auszugeben.

Mit der ASYNC-Anweisung unterstützt die Regelsprache einen asynchronen Datenfluß. Nicht angefragte Daten ermöglichen es zum Beispiel in einem System, das einen parallel arbeitenden Stratus Minicomputer aufweist, einer Regel auf dem Stratus, Daten zu einer Regel auf dem PC zu senden. Dies wird unterstützt unter Verwendung der folgenden Anweisungen:

- ASYNC ATTACH
- ASYNC DETACH
- ASYNC REFRESH
- ASYNC ROUTE

Der ASYNC ATTACH-Befehl startet das Empfangen von nicht angefragten Eingaben, z. B. gesendet von einer Regel auf dem Stratus, durch eine Regel auf dem PC. Der ASYNC DETACH-Befehl führt die exakte Umkehr von ATTACH aus. Er kann nur nach einem ATTACH-Befehl verwendet werden, wobei zu seinem Zeitpunkt der Empfang von nicht angefragten Eingaben unterbrochen wird. Der ASYNC ROUTE-Befehl schaltet eine Datenerneuerung von einer PC-Regel zu einer anderen. Die ASYNC REFRESH-Anweisung leitet das automatische Aktualisieren eines spezifischen Felds ein, sofern die ATTACH-Anweisung verwendet worden ist.

Die Regelsprache weist drei verschiedene Anweisungen auf, um den Steuerfluß innerhalb einer Regel zu ordnen.

Eine IF-Anweisung steuert die Ausführung basierend auf einer bestimmten Bedingung:

```
IF  Bedingung [ausführbare Anweisung ...]
[ELSE         [ausführbare Anweisung ...]  ]
ENDIF
```

Die CASEOF-Anweisung wählt einen von mehreren alternativen Ausführungswegen basierend auf dem Wert einer Variablen. Die Syntax ist

```
CASEOF  Variable
CASE  Konstante [ Konstante ... ] [ ausführbare Anweisung ...]
[ CASE  Konstante [ Konstante ... ] [ ausführbare Anweisung ...]
[ CASE OTHER ]
ENDCASE
```

Eine DO-Anweisung steuert die Ausführung von Wiederholungs-

schleifen. Die Syntax ist

```
DO [ FROM Datengröße ] [ TO Datengröße ] [ BY Datengröße ]
[ INDEX Variable ] [ ausführbare Anweisung ...]
[ WHILE Bedingung [ ausführbare Anweisung ... ]]
ENDDO
```

Alle Datengrößen und Variablen, die festgelegt werden, um eine DO-Schleife auszuführen, müssen ganze Zahlen sein. Die Vorbelegungen für die DO-Anweisungen sind: FROM = 1, TO = n, BY = 1, wobei n eine ganze Zahl ist.

Innerhalb der IF-, CASEOF- und DO-Anweisungen treten Bedingungen auf. Eine einfache Bedingung ist:

Ausdruck logischer Operator Ausdruck
oder Ausdruck INSET Set,

wobei der logische Operator einer der folgenden ist:

=	<=
<>	>
<	>=

Eine Bedingung ist aus einfachen Bedingungen oder anderen Bedingungen aufgebaut:

	einfache Bedingung
oder	(Bedingung)
oder	NOT Bedingung
oder	Bedingung AND Bedingung
oder	Bedingung OR Bedingung

Die INSET-Anweisung wird verwendet, um festzustellen, ob eine gegebene Variable oder Konstante als Wert innerhalb eines SET auftaucht. Zum Beispiel, gesetzt den Fall X ist ein Datengegenstand, dessen Datentyp kompatibel mit dem Datentyp des Set ist, dann ist:

```
X    INSET    SET NAME
```

eine Bedingung, die entweder TRUE oder FALSE ergibt, abhängig davon, ob mindestens einer der Werte in SET NAME den Wert von X trifft.

Ein Programmierer verwendet die Regelsprachenanweisungen, um die Logik des Programms auszuschreiben. Für weitere Informationen bezüglich der Regelsprache sollte auf die Anhänge A bis F, die hieran angehängt sind, Bezug genommen werden.

3. Benutzerschnittstellenfenster

Die Benutzerschnittstelle wird für jede Anwendung unter Verwendung des Fensterzeichners 16, Fig. 2 der CASE-Einrichtung konstruiert. Wie oben beschrieben, ist das Fensterzeichnermodul 16 das Programm, das vorgesehen ist, um in einem gegebenen Betriebssystem Benutzerschnittstellen aufzubauen. In dem Beispiel der obigen PC-Umgebung würde der Regelzeichner an das Microsoft Windows-Betriebssystem ankoppeln. Das Fensterzeichnermodul 16 würde dann die Maus-gestützte Benutzerschnittstelle aufweisen.

Das Fensterzeichnermodul 16 wird auch verwendet, um Bildschirmabbildungen von Daten zu erzeugen, die in den Feldelementen 102 und View-Elementen 100 (siehe Fig. 5) bezeichnet sind. Das Fensterzeichnermodul 16, Fig. 2, erzeugt eine Datei. Diese Datei enthält Daten zur Erzeugung einer Bildschirmabbildung des View-Fensters. Nach Erzeugen und Sichern der Maske erzeugt die CASE-Einrichtung der Erfindung auch andere Dateien, wie z. B. eine Datei, die den Code für die "gezeichnete" Maske enthält. Eine andere Datei könnte den Code für die Menüstrukturen enthalten, die von einer Maske verwendet werden.

G. Codeerzeugung

Nachdem ein Modell einer Anwendung gebildet worden ist, und die Regeln, Komponenten und Fenster erzeugt worden sind, stellt die CASE-Einrichtung einen Quellcode her.

Zubereitung ist die Codeerzeugungsphase. Wenn eine Anwendung zubereitet wird, werden die View-Elemente 100 und die SET-Elemente 116, Fig. 5, verwendet, um Kopierbücher zu erzeugen. Ein Kopierbuch ist eine Datei, die in den Quellcode eines Programmoduls hineinkopiert wird. Die CASE-Einrichtung erzeugt

ein Kopierbuch für jede Datenstruktur und schließt eine Kopie dieser Datei in jede ausführbare Regel, jedes Fenster bzw. Komponente ein, die sich auf einen bestimmten View oder Set beziehen.

Zubereitung erzeugt außerdem einen umgebungsspezifischen Quellcode für jedes Regelsprachenmodul. Das Regelelement 94, Fig. 5, enthält ein Attribut, das die Umgebungsbestimmung für jedes Regelmodul festlegt. Der Codeerzeuger 24, Fig. 2, nimmt die höheren Regelsprachenanweisungen und übersetzt sie in Quellcode in eine Sprache, die von der Hardwareumgebung der Bestimmung der Regel unterstützt wird. Zum Beispiel würde, falls eine Regel von einem Personal Computer auszuführen ist, der nur die C-Sprache unterstützt, der Codeerzeuger die Regelsprachenanweisungen in C übersetzen.

Falls die Regelsprachenanweisungen, die in einem Modul festgelegt sind, erfolgreich in einen Quellcode umgewandelt werden können, wird dieser Code in Dateien in dem Endlager 4 gespeichert. Falls es jedoch logische Fehler in den Regelsprachenanweisungen gibt, wird kein Code gesichert und Fehlermeldungen, die das erfolglose Ergebnis näher spezifizieren, werden in einer Fehlschlagdatei gesichert.

Nachdem die Zubereitung durchgeführt worden ist, können detaillierte Berichte über jedes Element in dem Elemente-Beziehungen-Modell gegeben werden. Diese Berichte sind die technische Dokumentation eines Programms.

Der Leser wird erneut auf Fig. 2A verwiesen, die ein Beispiel einer Umsetzung des Codeerzeugungsmoduls 24, Fig. 2, in der dreifach unterteilten Umgebung des Mainframe 1, des Minicomputers 2 und des PC 3 wiedergibt. Bei diesem Beispiel muß der Quellcode für jedes der Regelsprachenmodule nicht in der Hardwareumgebung erzeugt werden, die die Bestimmung jeder Regel sein wird; der Schritt des Kompilierens dieses Quellcodes würde jedoch in der Bestimmungsumgebung der entsprechenden Regel

erfolgen. In der PC-Umgebung 3 könnte eine Zubereitungseinrichtung 59, Fig. 2A, z. B. Quellcode in der Sprache C für jede vom PC ausführbare Regel unter Verwendung des Regelsprachencodes und alle View-Felder und Komponentenfenster, die zu diesem Regelelement des Regelmoduls zugehörig sind, zubereiten. Die Zubereitungseinrichtung 59 könnte ebenso z. B. COBOL-Quellcode für Regeln zubereiten, die zur Ausführung auf dem Mainframe bestimmt sind. In der Mainframe-Umgebung 1 stellt ein Mainframe Front-End-Modul 57 die Möglichkeit bereit, die Regelsprachenmodulanweisungen in Quellcodesprachenanweisungen zu übersetzen, die von der Hardwareumgebung der Bestimmung der Regel unterstützt werden. Das System 88 HPS Front-End-Modul 56 führt dieselben Aufgaben in einer Mikrocomputerumgebung aus.

Um bei dem Beispiel die Codeerzeugung durchzuführen, verwendet das Codeerzeugungsmodul in jeder Umgebung die Regelsprachenmodule plus Informationen, die von dem Elemente-Beziehungen-Modell benötigt werden. Um die Regelsprachenmodule zu transportieren und Quellcodemodule für die verschiedenen Umgebungen zu erzeugen, kann der Benutzer unter Verwendung des Datenbankadministratormoduls (DBA) 44 die Moduldateien in das zentrale Endlager 4 laden und dann verschiedene Module 34, 46 verwenden, um die Informationen zum Kompilieren in spezifische Umgebungen herunterzuladen.

In dem Fall einer Regel, die vorgesehen ist, von der PC-Umgebung 3 aus in der Mainframe-Umgebung 1 ausgeführt zu werden, stellt ein Haupt-View-Modul 36 über das Kommunikationsmanagermodul 8 Zugriff auf die Mainframe-Umgebung 1 bereit, wodurch ein nicht intelligenter Zugang in der PC-Umgebung emuliert wird.

Von der PC-Umgebung 3 wird in diesem Beispiel Zugriff auf die Mikrocomputerumgebung 2 über das View-Sprecher-Modul 46 bereitgestellt, daß mit dem System 88 Front-End-Modul 56 unter Verwendung eines PC-Verbindungsmoduls 50 und eines Dateiübersetzungsmoduls 48 kommuniziert. Auf dieser Stufe der Entwicklung können alle erzeugten Quellcodeanweisungen eines Moduls mit einem

Regel-View-Entstörer überprüft werden, der der jeweiligen Umgebung zugeordnet ist. Der Regel-View-Entstörer ist ein Aspekt der Testeinrichtung 10 der vorliegenden Erfindung und wird unten vollständiger beschrieben werden. Erfolgreich getestete Module können in dem zentralen Endlager 4 gespeichert werden.

I. Zusammenbau

Wenn alle Quellcodemodule einer Anwendung an die jeweilige Umgebung verteilt worden sind, müssen sie zu einem laufenden Computerprogramm zusammengebaut werden. Die Schritte sind: 1) Kompilieren des Quellcodes in Objektcode; 2) Errichten von Kommunikationsroutinen, um eine Interaktion der Programmodule über die Umgebungen hinweg zu ermöglichen; und 3) Binden bzw. Verknüpfen des kompilierten Codes.

Der Zusammenbau muß in jeder Umgebung ausgeführt werden, in die Programmodule verteilt wurden. Zum Beispiel wird bei einem System, das eine PC-Verarbeitung einschließt, der PC einen Systemzusammensetzer 28, Fig. 2, auf dem PC aufweisen, der es einem Programmierer ermöglicht, die Regeln auf dem PC zu kompilieren und zu binden.

In demselben Beispiel würde ein separater Zusammenbau auf einem Minicomputer abgeschlossen werden müssen. Zum Beispiel auf einem IBM S/88 oder Stratus Minicomputer würde ein Befehl wie "Build the Rule Router Application" die Grenzregeln nehmen und sie mit jedem anderen Regelmodul, das mit ihnen in der Mini-Umgebung in Beziehung steht, verknüpfen. Die Funktion bindet den Code auch zu einem ausführbaren Modul.

Mainframe Programmodule werden separat zusammengebaut. Zum Beispiel bei der beispielhaften IBM-Umgebung, die von einem CICS-Betriebssystem und einer DB2 relationalen Datenbank unterstützt wird, stellt ein Systemzusammenbauer 28, Fig. 2, auf dem Mainframe die Zusammenbaufunktionen für den Regel- und Komponentenquellcode bereit. Bei einer Regel würde ein Zusammenbauer:

- verifizieren, daß die Regel eine gültige Mainframe-Regel ist;
- verifizieren, daß die Regel den Codeerzeugungsprozeß abgeschlossen hat;
- den Benutzer abfragen, um anzuzeigen, ob die Regel als Grenzregel aufzubauen ist;
- die Bindungs-Datei lesen, um die On-line-Views-Datei und die On-line-Beziehungen-Datei für die Laufzeit-PCI- und Umwandlungsverwendung zu laden;
- den Quellcode in die On-line-Quelldatei für die Regel-View-Verwendung laden;
- automatisch die DB2-Bindung ausführen, basierend auf den Beziehungen, die gegenüber dem Endlager für Grenzregeln festgelegt sind, welche die DB2-Aufrufe ausgebende Komponenten verwenden;
- eine Liste von anderen Regeln und Komponenten anzeigen, die von dem Aufbau einer bestimmten Regel betroffen sind;
- den Grenzregeln, die DB2-Aufrufe ausgebende Komponenten verwenden, eine einheitliche Bezeichnung und einen einheitlichen DB2-Plan-Namen zuordnen;
- einen DB2-Plan für eine Grenzregel neu binden, wenn eine der betroffenen Komponenten der DB2-Aufrufe ausgebenden Regel, verändert worden ist;
- die Regel aus der Mainframe-Umgebung entfernen, wenn die Regel nicht länger benötigt wird.

Ein Komponentenzusammenbauer in einer Mainframe-Umgebung würde diese Funktionen ausführen:

- einem Programmierer die Einrichtungen zur Verfügung stellen, um seine Komponente in eine von dem System unterstützte Sprache zu kompilieren;
- dem Programmierer erlauben, die Ergebnisse seines Kompilierens einzusehen (Kompilierungslisten und editierte Verknüpfungsabbildungen);
- eine Liste von anderen Regeln und Komponenten anzeigen, die von dem Aufbau der speziellen Komponente berührt werden;
- dem Benutzer erlauben, den Quellcode einer Mainframe-

Komponente zu editieren.

J. Ausführung

Nach dem erfolgreichen Zusammenbau der Anwendungsmodule in jeder Hardwareumgebung kann das Programm jetzt ausgeführt und getestet werden. Die CASE-Einrichtung stellt die Möglichkeit bereit, die Anwendung von jeder Umgebung in der Hardwarearchitektur aus auszuführen und zu testen. Zum Beispiel hätte ein Programmierer bei Verwendung der Hardwarearchitektur bestehend aus einem IBM Mainframe, einem IBM Stratus Minicomputer und PC-Arbeitsplätzen, wie in Fig. 1, an einem PC sitzend die Optionen:

- die PC-gestützten Teile einer Anwendung auszuführen, wobei keine Verknüpfungen mit Modulen, die in der Stratus- oder Mainframe-Umgebung ausgeführt werden, erzeugt werden;
- die Module und die modulare Ausführung in anderen Umgebungen zu testen. In diesem Fall werden Verknüpfungen mit Anschlüssen anderer Umgebungen erzeugt. Dies erlaubt Kommunikation zwischen dem Umgebungen;
- das gesamte Programm auszuführen bzw. zu testen. In diesem Fall werden Verknüpfungen mit jeder der erforderlichen Ausführungsumgebungen erzeugt.

K. Überarbeiten einer Anwendung

Der Prozeß des Änderns der Elemente bei einer Anwendung unter Verwendung des Elemente-Beziehungen-Modelliersystems und der Regelsprache ist ein geradliniger Prozeß. Es wird in die Datenbank eingetreten, und das Programmelement wird geändert. Kleine Änderungen in diesen Systemen können jedoch große Konsequenzen haben. Grundsätzlich müssen alle Elementtypen, die Elemente, die geändert worden sind, besitzen, verwenden oder einschließen, neu zubereitet bzw. "modifiziert" werden. Die CASE-Einrichtung führt diese Modifikation durch das Softwareverteilungssystem 32, Fig. 2, durch. (Siehe die vorgenannte internationale Anmeldung, die durch Bezugnahme inkorporiert wurde.)

L. Testeinrichtung

Letztlich stellt das CASE-System eine Entstör- und Testeinrichtung bereit, die es dem Benutzer ermöglicht, die Leistungsfähigkeit einer Anwendung zu ermitteln.

Allgemein werden die Anwendungen zuerst in den Computer auf Codeniveau eingegeben. Traditionelle Codeentstör-Tools sind zum Testen ausschließlich auf dem Codeniveau vorgesehen. Anstelle dessen stellt die CASE-Einrichtung der vorliegenden Erfindung ein Regel-View-Modul für ein höheres Entstören bereit, was Teil der Testeinrichtung 10, dargestellt in Fig. 1, ist. Eine Version des Regel-View würde bei einer Hardwarekonfiguration in jeder der Betriebsumgebungen existieren. Anwendungen, die mehr als eine Umgebung überspannen, erfordern ein separates Testen der Module in jeder Umgebung.

Anrufe des Regel-View-Moduls werden von der CASE-Einrichtung automatisch eingebettet, wenn Code erzeugt wird. Der Regel-View läßt die Anwendung in einem Entstörer auf Regelniveau laufen, der auftretende Fehler und Probleme lokalisiert. Er kann interaktiv verwendet werden, um einen Regelprozeß durchzusehen und um die Inhalte eines View-Kopierbuchs an jedem Punkt zu überprüfen. Der Regel-View gibt Programmierern die Möglichkeit:

1. die Ausführung einer Regel einzuleiten;
2. die Schnittstelle zwischen einer Regel und einer zweiten Regel oder Komponente zu unterbrechen;
3. in die Logik einer Regel einzutreten;
4. ein Regel- oder Komponentenmodul durchzugehen;
5. an den Beginn eines Moduls zurückzugehen;
6. jedes Feld innerhalb eines View, der von der aktiven Regel besessen wird, zu überprüfen und zu modifizieren;
7. den Quellcode der aktiven Regel durchzusehen;
8. jegliche View-Daten für zukünftige Bezugnahme und Wiederverwendung zu sichern;
9. den Regelquellcode und View-Daten auszudrucken.

Der Regel-View wird an jedem Unterbrechungspunkt eine Liste

aller Views, die überprüft und editiert werden können, zusammenstellen. Die Anzahl der angezeigten Views hängt davon ab, wo und wie der Regel-View verwendet wird. In den meisten Fällen werden die Views, die zur Anzeige verfügbar sind, die Eingabe und Ausgabe der ausgeführten Regel enthalten. In weiter fortgeschrittenen Situationen -- wenn asynchrone Regeln entstört werden oder mehrfache Anwendungen getestet werden -- können viele Views zur Überprüfung verfügbar sein. Die Inhalte eines Views können ausgedruckt oder in einer Datei gesichert werden.

Der Regel-View-Editor erlaubt es dem Programmierer, jegliche Daten durch Verwendung eines Feldeditors zu editieren. Mit dem Feldeditor kann ein Programmierer jegliche Daten durch Überschreiben der alten Informationen ändern. Der Editor erlaubt dem Benutzer außerdem, die Inhalte jedes Felds in ihre Originalwerte zu restaurieren.

Um einen Quellcode einer Regel zu revidieren, stellt ein Texteditor eine Option zur Verfügung, in der die Zeilen des Quellcodes, die aktuell ausgeführt wird, zu jeder Zeit hervorgehoben wird. Falls mehrere Module unter dem Regel-View laufen, ist der angezeigte Quellcode der Quellcode, der die Daten in dem aktuellen View besitzt.

M. Softwareverteilung für allgemeine Anwendung

Die vorangegangene Diskussion der CASE-Tool-Einrichtung der vorliegenden Erfindung war auf das Modellieren und die Entwicklung einer einzigen Kopie eines Anwendungsprogramms beschränkt, das beispielsweise über drei Hardwareumgebungen eines PC, eines Mikrocomputers und eines Mainframe verteilt wurde. Eine einzige Kopie der Module, die das Programm ausbilden, wurde an die geeigneten Hardwareumgebungen verteilt, und die Anwendung wurde entstört, und ihre Leistungsfähigkeit wurde beurteilt. Nachdem die Anwendung jedoch getestet und fertig für die allgemeine Anwendung ist, müssen viele Kopien der das Programm ausbildenden Module verteilt und gepflegt werden. Das Softwareverteilungs-

modul 32 der vorliegenden Erfindung wird eine Vielzahl von Kopien der Module einer Anwendung über eine parallelverarbeitende Hardwareumgebung verteilen, so daß viele Benutzer dasselbe Programm benutzen können. Für mehr Informationen über das Softwareverteilungssystem siehe die vorgenannte internationale Anmeldung.

N. Umwandlung für nicht-intelligente Endgeräte (NITC)

Zusätzlich zu der oben erwähnten Hardwarekonfiguration kann die CASE-Einrichtung der vorliegenden Erfindung auch unter Verwendung einer Hardwarekonfiguration umgesetzt werden, bei der der Endbenutzer des Anwendungsprogramms keinen Zugang zu einem verarbeitenden Endgerät, wie beispielsweise einem PC, hat. Stattdessen würde der Endbenutzer nur Zugang zu einem nicht-intelligenten Endgerät haben. Ein Beispiel wäre ein Terminal 3270 der Marke IBM, der an einen Mainframe Computer wie den IBM 3090 angeschlossen ist, der oben als beispielhafte Ausführungsform beschrieben wurde. Es können aber auch andere geeignete Endgeräte benutzt werden, wenn die Erfindung in anderen Mainframe-Hardwareumgebungen umgesetzt wird. Ein nicht-intelligentes Endgerät wird manchmal verwendet, weil es weniger teuer für große Benutzergruppen beim Zugriff auf einen einzigen Mainframe Prozessor ist, als jedem Benutzer seinen oder ihren eigenen Personal Prozessor zu geben. In diesen Fällen kann es jedoch sein, daß die Programmierspezialisten weiterhin Anwendungen mit einem PC entwickeln unter Verwendung derselben Entwicklungswerkbankmodule 34, wie sie in Fig. 2A beschrieben sind, entwickeln würden. Diese Entwicklungs-Tools erlauben es dem Programmierer zum Beispiel, Benutzerschnittstellendateien (beispielsweise Fensterdateien) aufzubauen, die in Verbindung mit der oben beschriebenen Umwandlungsfunktion der CASE-Einrichtung verwendet werden. Vorausgesetzt, daß der Endbenutzer in dem obigen Fall keinen PC hat, um die erzeugten Fenster umzuwandeln, wäre die grafische Schnittstelle, die in den Fensterdateien beschrieben ist, sinnlos.

Die vorliegende Erfindung stellt dennoch eine Möglichkeit bereit, unter Verwendung eines Umwandlungsmoduls für nicht-intelligente Endgeräte (NITC) *140*, Fig. 2A, eine grafische Schnittstelle zu erzeugen. Das Umwandlungsmodul für nicht-intelligente Endgeräte macht es möglich, Anwendungen aufzubauen, die einem Endbenutzer, der beispielsweise eine nicht-intelligente, an einen Mainframe angeschlossene Vorrichtung verwendet, grafische Benutzerschnittstellen anzeigen. Das Umwandlungsmodul für nicht-intelligente Endgeräte kann auch bestehende PC-Arbeitsplatz-Softwaremodule umwandeln, so daß sie in einer Mainframe-Umgebung ausgeführt werden können. Durch Bereitstellen dieser Funktion der vorliegenden Erfindung kann unter Verwendung der Entwicklungswerkbankmodule 34 eine auf dem Mainframe ausgeführte Anwendung vollständig innerhalb einer PC-Umgebung entworfen, aufgebaut, entstört und als Prototyp fertiggestellt werden, ohne vor der öffentlichen Verteilung auf den Mainframe zuzugreifen.

In dem Elemente-Beziehungen-Modell ist das Fensterelement weiterhin für ein Fensterelement definiert, das für eine grafische Benutzerschnittstelle auf dem Mainframe verwendet wird.

In der PC-Umgebung ermöglichen es die Entwicklungswerkbankmodule 34 einem Benutzer, die Fensterobjekte (Masken genannt) zu zeichnen, mit denen der Endbenutzer der Anwendung interagiert. Das Fensterzeichnermodul wurde oben diskutiert.

Die Maske kann als zu dem Fensterelement zugehörige Datei in dem zentralen Endlager 4 gespeichert werden. Diese Masken müssen jedoch unter Verwendung von Bildschirmobjekten, die von der Endgeräteumwandlungsvorrichtung für nicht-intelligente Endgeräte unterstützt werden können, aufgebaut werden.

Ein Fenster zur Ausführung sowohl auf dem PC und dem nicht-intelligenten Endgerät kann z. B. durch Auswählen einer Option in dem Fensterzeichnermodul 16 spezifiziert werden. Die Anwahl

der Option wandelt das aktuelle Fenster in ein Format um, das den Umwandlungsbildschirm des nicht-intelligenten Endgeräts imitiert. Zum Beispiel könnten Felder in Reihen-/Zeilen-Koordinaten übersetzt werden, Drucktasten könnten in die Fußzeile des Schirms verschoben werden und Tastaturäquivalente fordern. Das Auswählen dieser Anwahl versetzt außerdem das Fensterzeichnermodul in einen Maskenaufbaumodus für nicht-intelligente Endgeräte, der die Anordnung von neuen Objekten verhindert, die nicht von einer Endgeräteumwandlungsvorrichtung für nicht-intelligente Endgeräte unterstützt werden können.

Der Hauptunterschied zwischen einem PC-Arbeitsplatz und einer nicht-intelligenten Endgerätvorrichtung ist, daß die Umwandlungsvorrichtung für nicht-intelligente Endgeräte Textzeichen manipuliert, während der PC-Arbeitsplatz grafische Pixel manipuliert, die dem PC-Arbeitsplatz ein höheres Maß an Steuerfunktion verleihen. Um diesem Unterschied Rechnung zu tragen, könnte das Fensterzeichnermodul z. B. sämtliche Textobjekte in einen Standardzeichensatz umwandeln.

Die in der Mainframe-Umgebung erzeugten Fenster unterstützen viele der Möglichkeiten, die bei PC-Umgebungs-Fenstern vorhanden sind, einschließlich beispielsweise Farben, editierbare Felder und Textfelder (beispielsweise nur zur Anzeige), Feldanzeige-attribute (beispielsweise Größe oder Art des Felds, Bereich usw.), Drucktasten und andere Funktionsschlüsseläquivalente.

Dem Aufbau der Anwendung auf den PC-Arbeitsplatz folgend, müssen die die Anwendung ausmachenden Objekte, in den Mainframe geladen werden, um für die Ausführung zubereitet zu werden. Der Laufzeitanteil des Umwandlungsmoduls für nicht-intelligente Endgeräte managt alle Umwandlungsfunktionen und stellt die Einrichtungen für einen Entwickler bereit, um die Umwandlungs-funktionen dynamisch zu verändern.

Die vornehmliche Funktion der Umwandlungsmodule ist es, die gesamte Eingabe und Ausgabe über den Bildschirm des nicht-

intelligenten Endgeräts für ein Fenster umwandelnde Regel zu managen. Zusätzlich teilt das Umwandlungsmodul für nicht-intelligente Endgeräte die Aktionen des Endbenutzers der Regel durch Besetzen eines ausgewählten Felds mit der ausgewählten Textfolge mit. Bevor die Eingabe des Benutzers an die aufrufende Regel übertragen wird, führt das Umwandlungsmodul die notwendige Editierung und Validierung jedes Eingabefeld aus, wie sie durch die in dem Endlager und der Fensterdefinition definierten Attribute festgelegt ist.

Das Umwandlungsmodul für nicht-intelligente Endgeräte wird jedesmal aufgerufen, wenn eine Regel ein Fenster umwandelt. Wenn eine Regel ein Fenster umwandelt, liest das Umwandlungsmodul für nicht-intelligente Endgeräte eine vorbestimmte Datei ein, um die physikalischen Attribute des Fensters zu erhalten. Diese physikalischen Attribute werden verwendet, um die Datenstrombefehle aufzubauen, die verwendet werden, um das Fenster auf einer nicht-intelligenten Endgerätvorrichtung anzuzeigen.

Das Umwandlungsmodul für nicht-intelligente Endgeräte führt eine Editierung und Validierung von allen Eingabefeldern aus, bevor die Steuerung zu der aufrufenden Regel zurückkehrt. Bei allen numerischen Feldern stellt das Umwandlungsmodul für nicht-intelligente Endgeräte sicher, daß die eingegebenen Daten numerisch sind und stellt die Stellenwertigkeit basierend auf einem eingegebenen Dezimalpunkt oder dem implizierten Dezimalpunkt, der in einer numerischen Bildfolge festgelegt ist, ein.

Zum Beispiel wird eine in das Feld eingegebene Folge 123, falls ein Dezimalfeld als fünf Stellen lang mit zwei der Stellen rechts von dem Dezimalpunkt definiert und unter Verwendung einer numerischen Bildfolge "9999.9" editiert ist, die folgenden Resultate haben. Das Dezimalfeld wird den Wert 12.30 enthalten und das Bildschirmfeld wird 0012.3 anzeigen. Zusätzlich zur Verwendung numerischer Bildfolgen zum Editieren und Validieren numerischer Eingaben kann unter Verwendung des Fensterzeichnermoduls eine Bereichsgrenze gesetzt werden, um sicherzustellen,

daß der eingegebene numerische Wert nicht jenseits der Minimum- und Maximum-Werte des festgelegten Bereichs liegt.

Buchstabenfelder werden nur validiert, falls der Eingabewert mit einem Set von zulässigen Werten übereinstimmt. Dieser Set und seine Werte werden als Elemente gegenüber dem zentralen Endlager definiert. Um bei einem Feld festzulegen, daß es einen Set von zulässigen Werten anfordert, muß der Name des Setelements in dem Referenzdateiattribut des Felds festgelegt sein, wenn die Objekteditierfestlegungen des Felds unter Verwendung des Fensterzeichnermoduls getroffen werden. Das Umwandlungsmodul für nicht-intelligente Endgeräte wird verifizieren, daß die eingegebene Buchstabenfolge in dem Set der zulässigen Werte enthalten ist.

Bei jeglichen fehlerhaften Feldern wird das Umwandlungsmodul für nicht-intelligente Endgeräte die fehlerhaften Felder markieren und eine Fehlermeldung anzeigen. Bei nicht-intelligenten Vorrichtungen, die Farbe unterstützen, können die fehlerhaften Felder hervorgehoben werden.

Wie früher festgestellt, können die Drucktastenobjekte auf einem PC-Arbeitsplatz-Fenster als Funktionstasten auf der nicht-intelligenten Endgerätvorrichtung umgesetzt werden. Jede Funktionstaste, die für ein Fenster definiert ist, hat einen ihr zugehörigen Textwert. Dieser Textwert wird von dem Umwandlungsmodul für nicht-intelligente Endgeräte an die aufrufende Regel in ein bestimmtes Feld zurückgegeben.

Die oben beschriebene Ausführungsform der Erfindung ist nur zur Erläuterung vorgesehen, da bestimmte Änderungen an ihr ausgeführt werden können, ohne von der klaren Lehre der Erfindung abzuweichen. Dementsprechend ist auf die nachfolgenden Ansprüche zu verweisen werden, die die Erfindung vornehmlich definieren.

Application No.: 91 901 023.1
Applicant: Seer Technologies, Inc.
8000 Regency Parkway
US - Cary, North Carolina 27511

ANHANG A: REGELSPRACHENSYNTAX

Token der Regelsprache

Token sind die Atome, aus denen eine Programmiersprache aufgebaut ist. Alle reservierten Wörter, so wie IF, AND, FROM, ..., stehen für Token der Regelsprache. Andere Beispiele sind spezielle Symbole wie ")" (rechte Klammer), "<=" (weniger oder gleich-Verhältnisoperator) und dergleichen. Letztlich wird man auch Token, so wie DICT_View (View-Name), begegnen, die Bezugsquellen aus dem Endlager bezeichnen.

Reservierte Worte

ABS	EVERY	OVERLAY
ALL	EXISTS	PC
AND	EXP, EXP 10	PREV
ASCENDING	EXTERN	PUT
ASYNCH	EXTRACT	QUEUE
ATTACH	FALSE	REFRESH
AVG	FIELD	RETRIEVE
BEEP	FLASH	RETURN
BY	FORALL	RIGHTJ
CASE	FROM	ROUND
CHAR	IF	ROUTE
CICS	IN	RULE
CLEAR	INDEX	SET
COMPONENT	INSET	SETERROR
CONVERSE	INTEGER	SMALLINT
CURRENT	LEFTJ	SPACES
DCL	LENGTH	SQL
DELETE	LIKE	SQRT
DEPENDING	LOG, LOG10	STRATUS
DESCENDING	MAP	STRING
DETACH	MAX	SUBSTRING
DIV	MIN	SUM
DO	MOD	TO
DOMAIN	MODULE	TRIM
ELSE	MOVE	TRUE
EMPTY	NEST	TYPE
ENDCASE	NEXT	USE
ENDDCL	NOT	VIA
ENDDO	NUMERIC	VIEW
ENDEXTERN	OCCUR	WHILE
ENDFORALL	OF	WINDOW
ENDIF	ON	ZERO
ENDSET	OR	ZEROES

Reservierte Symbole

Zeichen	Beschreibung	Symbol
*>	linke Bemerkung	-
<*	rechte Bemerkung	-
(	linke Klammer	LP
)	rechte Klammer	RP
=	gleich	EQ
>=	größer oder gleich	GE
>	größer als	GT
<=	kleiner oder gleich	LE
<	kleiner als	LT
<>	nicht gleich	NE
,	Komma	COMMA
;	Semikolon	SEMI

Endlagerelementbezeichnungen

MODULE_NAME
RULE_NAME
WINDOW_NAME
VIEW_NAME
SYMBOL_NAME
SET_NAME

Vorrangtabelle

Die folgende Vorrangtabelle listet den Vorrang bzw. die Bindungswirkung der Regelsprachenoperatoren in ansteigender Ordnung auf. Operatoren, die in derselben Zeile auftreten, haben untereinander dieselbe Bindungswirkung.

OR	links nach rechts
AND	links nach rechts
NOT	rechts nach links
EQ NE LE LT GE GT	nicht-assoziativ
INSET	nicht-assoziativ
MINUS	links nach rechts
IN	nicht-assoziativ
OF	rechts nach links

The rechte Spalte beschreibt die Assoziativität der Operationen. Aus der "AND"-Zeile der vorstehenden Tabelle ist zu entnehmen, daß es völlig zulässig ist, für Bedingungen cond1, cond2, cond3 zu schreiben

```
cond1    AND  cond2    OR   cond3
```

und daß die implizierte Ordnung bzw. der syntaktische Aufbau zuerst

```
cond1    AND  cond2
```

und anschließend

cond2 AND cond3

ist.

Anderes Beispiel: Gegeben sei eine teilweise Qualifikation

V1 OF V2 OF V3

die drei Views V1, V2, V3 einbezieht, dann erkennt der syntaktische Analysierer zuerst

V2 OF V3

und anschließend

V1 OF V2.

ANHANG B: REGELSPRACHENGRAMMATIK (BNF)

```
rule_code:          dcl_s_list  stmt_list

dcl_s-list:         empty
                  | dcl_s_list   dcl_stmt

dcl_stmt:           DCL  dcl_1st  ENDDCL

dcl_1st:            dcl_item
                  | dcl_lst   dcl_item

dcl_item:           dcl_idx_item
                  | dcl_chr_item
                  | dcl_int_item
                  | dcl_like_item
                  | dcl_ext_item

dcl_idx-item:       dclvar_list   INDEX SEMI

dcl_chr_item:       dclvar_list chrtype SEMI

dcl_int_item:       dclvar_list inttype SEMI

dcl_like_item:      dclvar_list liketype SEMI

dcl_ext_item:       extern dclvar_list decttype SEMI

dclvar_list:        dclvar_item
                  | dclvar_list  COMMA  dclvar_item

dclvar_item:        simple_var
                  | simple_var  dcl_subscr

chrtype:            CHAR
                  | CHAR dcl_subscr

inttype:            SMALLINT
                  | INTEGER

liketype:           LIKE simple_var

dcl_subscr:         LP  int_lit  RP

dicttype:           SET

stmt_list:          (empty)
                  | stmt_list  stmt

stmt:               assign_stmt
                  | overlay-stmt
                  | clear stmt
                  | use stmt
                  | return_stmt
                  | converse_stmt
                  | async_stmt
                  | cond_stmt
                  | do_stmt
```

```
        |           do_idx_stmt

assign_stmt:        MAP expr TO variable

overlay_stmt:       OVERLAY dat_item TO variable

clear_stmt:         CLEAR variable

use_stmt:           USE MODULE MODULE_NAME
        |           USE RULE RULE_NAME nest clause

nest_clause:        (empty)
        |           NEST

return_stmt:        RETURN

converse_stmt:      CONVERSE window_clause

window_clause:      WINDOW WINDOW_NAME

async_stmt:         ASYNC attach_detach RULE_NAME VIA RULE
                    RULE_NAME
        |           ASYNC ROUTE RULE RULE_NAME TO RULE
                    RULE_NAME
        |           ASYNC refresh_stmt

attach_detach:      ATTACH
        |           DETACH

refresh_stmt        REFRESH window_clause
                    field_clause
                    view_clause
                    occur_clause
                    beep_clause
                    flash_clause

view_clause:        (empty)
        |           VIEW data_item

field_clause:       (empty) clause
        |           FIELD data_item

occur clause:       (empty)
        |           OCCUR data_item

beep-clause:        (empty)
        |           BEEP

flash_clause:       (empty)
        |           FLASH

cond_stmt:          if_stmt
        |           case_of-stmt
        ;
```

```
if-stmt:            IF cond stmt-list ENDII
        |           IF cond stmt_list  ELSE stmt_list ENDIF

case_of_stmt:       CASE_OF caseof_var case_list  ENDCASE
        |           CASE_OF caseof_var case_list  CASE OTHER
                    stmt_list ENDCASE

caseof_var:         variable

case_list:          single_case
        |           case_list single_case

single_case:        CASE case_lit_list  stmt_list

case_lit_list:      lit
        |           set_const
        |           MINUS int_lit
        |           MINUS dec_lit
        |           case_lit_list lit
        |           case_lit_list  set_const

do_stmt:            DO stmt_list  WHILE cond stmt_list ENDDO
        |           DO stmt_list ENDDO

do_idx_stmt:        DO do_clauses stmt_list while_clause ENDDO

do_clauses:         from_clause to_clause by_clause
                    index_clause

from_clause:        (empty)
        |           FROM expr

to-clause:
        |           TO expr

by_clause:
        |           BY expr

index_clause:
        |           INDEX variable

while_clause:
        |           WHILE cond stmt_list

cond:               simple_cond
        |           LP cond RP
        |           not cond %prec NOT
        |           cond AND cond %prec AND
        |           cond OR cond %prec OR

simple_cond:        expr  rel_op   expr
        |           expr   INSET SET_NAME

rel_op:             EQ
        |           NE
        |           num_rel_op
```

```
num_rel_op:         LT
                    LE
                    GT
                    GE

variable:           simple-var
                    qual-id qual_var_list
                    simple_var subscr_unit
                    qual_id qual_var_list subscr_unit

subscr_unit:        LP item_list RP

item_list:          expr
                    expr COMMA expr
                    expr COMMA expr COMMA expr

qual_var_list:      OF VIEW_NAME
                    qual_var_list OF VIEW_NAME

dat_item:           variable
                    lit
                    set_const

set_const:          SYMBOL_NAME
                    SYMBOL_NAME IN SET_NAME

lit:                char_lit
                    int_lit
                    dec_lit
```

ANHANG C: REGELSPRACHENSEMANTIK

Bezeichnungen

Bezeichnungen werden verwendet, um bestimmte Sprachkonstrukte zu benennen, so wie Variablen (z. B. Felder und Views) und andere Endlagerelemente, so wie Regeln und Komponenten.

Eine Feldbezeichnung muß mit einem alphabetischen Zeichen beginnen. Ihm folgt optional eine Sequenz von einem oder mehreren Buchstaben, Zahlen oder Unterstrichen. Zwischen Großschreibung und Kleinschreibung wird kein Unterschied gemacht; deshalb benennen eine_lange_bezeichnung und EINE_LANGE_BEZEICHNUNG dieselbe Sache, soweit die Regelsprache betroffen ist. Eine Bezeichnung kann keine "weißen Felder" enthalten (Leertasten oder Tabulatoren oder neue Zeilen).

Man kann Namen für Bezeichnungen verwenden, die reservierte Worte innerhalb der Sprache COBOL oder C oder jeder anderen Sprache sind (so wie SENTENCE oder SIZE). Es ist jedoch zu beachten, daß bestimmte Worte als Schlüsselworte oder für zukünftige Erweiterungen der Regelsprache reserviert worden sind (siehe Anhang A). Außerdem gelten für die Bezeichnungen Beschränkungen bezüglich ihrer Länge, abhängig davon, was sie benennen. Diese Beschränkungen sind in Fig. 1 aufgelistet.

Bezeichnungstyp	Maximale Anzahl der Zeichen
Feld	18
View	8
Symbol	18
Set	8
Fenster	8
Regel	7
Komponente	7

Fig. 1 Längenbeschränkungen der Bezeichnungen

Anhang A enthält eine Liste von "reservierten Worten" für die Regelsprache. Diese Bezeichnungen können nicht verwendet werden, um irgendein anderes Sprachenkonstrukt zu benennen, so wie Felder, Views, Regeln, Komponenten usw..

Operatoren und Begrenzer

Die folgenden Zeichen haben besondere Bedeutung für die Regelsprache und werden als binäre und unäre Operatoren verwendet.

Symbol	Name	Funktion
(	linke Klammer	Gruppierung;
)	rechte Klammer	Tiefindizes
=	Gleichheitszeichen	zum Bilden von Symbolen oder Verhältnisoperatoren
<	kleiner als-Zeichen	
>	größer als-Zeichen	
<>	ungleich-Zeichen	Test auf Ungleichheit
<=	kleiner als- oder gleich-Zeichen	Test auf kleiner als oder gleich
>=	größer als- oder gleich-Zeichen	Test auf größer als oder gleich
-	Minuszeichen	Verneinung
,	Komma	Trennung von Listeneelementen und Tiefindizes
;	Semikolon	Beendigung einer Erklärung eines lokalen Felds
+	Addition	zur zukünftigen Verwendung
*	Multiplikation	" " "
/	Division	" " "
++	Vereinigung	" " "
**	Unterteilung	" " "
--	Satzunterschied	" " "
<<	Untersatz	" " "
>>	Übersatz	" " "

Fig. 2 Binäre und unäre Regelsprachenoperatoren

Felder
Ein Feld ist eine Variable, die sich wie ein Atom verhält; das heißt, sie kann nicht in kleinere Einheiten unterteilt werden. Mit der Ausnahme einer begrenzten Möglichkeit zur lokalen Erklärung von Feldern gegenüber einem Regelprogramm werden Felder unter Verwendung der Einrichtungen des HPS-Endlagers definiert. Sie werden nicht innerhalb einer Regel definiert. Einige Beispiele von Feldern und deren Typen folgen.

```
FLD_1                   SMALLINT

WORD_COUNT              INTEGER
```

In dem oben gegebenen Beispiel ist FLD_1 eine zwei Byte (relative) ganzzahlige Variable, und WORD_COUNT ist eine vier Byte (relative) ganzzahlige Variable.

```
FLD_2                   CHAR

CUSIP_DESCR             CHAR (20)
```

FLD_ und CUSIP_DESCR sind Zeichenfelder der Längen 1 bzw. 20.

```
CUST_BAL_AMT            DECIMAL (15, 2)
```

CUST_BAL_AMT ist eine "Dollars und Cents"-Variable mit bis zu 15 Stellen Genauigkeit, von denen zwei auf der rechten Seite eines implizierten Dezimalpunkts angeordnet sind. DECIMAL (p,q)-Felder sind relative Größen.

```
CASH_TXN_BAL_AMT        PIC S9999V99
```

CASH_TXN_BAL_AMT und CASH_TXN_CR_AMT sind beides numerische Variablen mit bis zu 4+2 = 6 Stellen Genauigkeit, von denen zwei auf der rechten Seite eines implizierten (V="virtuellen") Dezimalpunkts angeordnet sind. Das S in dem ersten Beispiel bezeichnet ein optionales Vorzeichen, während in der Erklärung:

```
CASH_TXN_CR_AMT         PIC 9999V99
```

CASH_TXN_CR_AMT nicht negativ sein kann.

```
TEMP_BUFFER             VARCHAR (50)
```

Die Konstrukte der Regelsprache unterscheiden einen TEMP_BUFFER nicht von einer CHAR (50) Variablen. Sowohl CHAR (50) als auch VARCHAR (50) ordnen 50 Byte Speicher zu, und in beiden Fällen wird alles, was in einen von ihnen eingebracht wird, linksbündig angeordnet und nach rechts mit Freiräumen aufgefüllt. Eine VARCHAR (n) Variable hält die Spur ihrer "aktuellen" Länge durch Verwendung eines separaten Längenfelds, welches für den Regelsprachenprogrammierer vollständig transparent ist.

Die folgende Tabelle faßt die Feldtypen zusammen, die in der HPS-Regelsprache unterstützt werden, und einige ihrer Eigenschaften:

Bemerkungen

Bemerkungen sind zwischen "*<" und "*>" als Begrenzungen eingeschlossen. Bemerkungen können nicht verschachtelt werden. Es gibt keine Grenze für die Länge einer Bemerkung.

```
*>   Dies ist ein Beispiel einer Bemerkung <*
*>   Dies ist <* *> ein anderes Beispiel <**>
     einer Bemerkung <*

*>

*    es ist
*    zulässig,
*    eine Bemerkung
*    über mehr als
*    eine Zeile zu verteilen
*    und * oder < in Alleinstellung zu verwenden
<*
```

Eingabe nach Spalte 72 wird als Bemerkung behandelt (d. h. ignoriert).

Datengrößen

Datengrößen sind die Variablen und Konstanten in der Regelsprache. Fig. 3 zeigt, wie diese Datengrößen in Beziehung zueinander stehen.

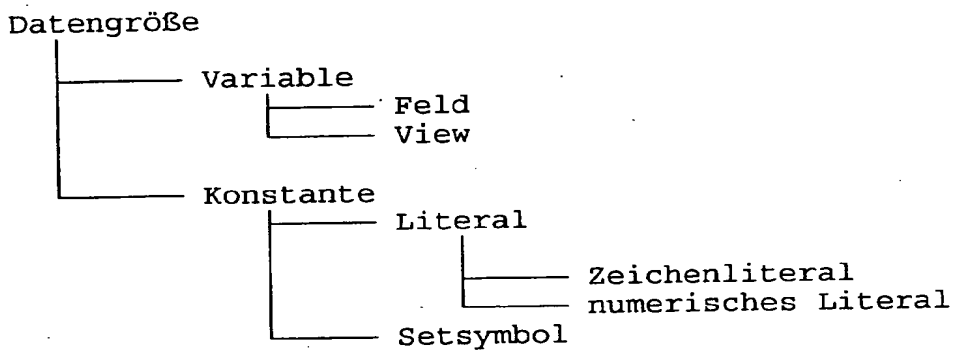

Fig. 3 Datengrößenhierarchietabelle

Variablen

Variablen werden definiert unter Verwendung der View- und Feld-HPS-Elemente. Felder und Views werden unter Verwendung der Einrichtungen des Endlagers außerhalb der Sprache selbst definiert und beschrieben. Es ist zu anzumerken, daß mit bestimmten Beschränkungen, Felder und Views mittels des DCL-Konstrukts lokal gegenüber dem Regelblock erklärt werden können!

Typ	Darstellung (1)	Zeichen/ Nummer	Anmerkung
Zeichen	CHAR (nn)	Zeichen	(2)
Zeichenvariable	VARCHAR (nn)	Zeichen	(3)
2-Byte Ganzzahl	SMALLINT	Zahl	(4)
4-Byte Ganzzahl	INTEGER	Zahl	(4)
Dezimalzahl	DECIMAL (p,q)	Zahl	(4)
relatives Bild	PIC S9999999v9999999 p Anzahl q Anzahl	beides	(5,6)

Anmerkungen:

(1) "Darstellung" bezieht sich auf das Format der DCL ... ENDDCL-Anweisungen für eine Variable jenen Typs. Es ist anzumerken, daß die aktuelle Version von HPS lokale Erklärungen für die Datentypen CHARACTER, VARCHAR und sowohl SMALLINT und INTEGER Variablen unterstützt. DECIMAL wird nicht unterstützt. Siehe Abschnitt 4 (Feldelementtypen) für weitere Informationen bezüglich der Datentypen.

(2) CHAR ist das gleiche wie CHAR (1).

(3) VARCHAR ist das gleiche wie VARCHAR (1).

(4) Ganzzahl- und Dezimalzahlvariablen sind relative Variablen.

(5) Relative Bildvariablen sind relativ, nicht relative Bildvariablen sind nicht relativ.

(6) Bildvariablen sind von doppelter Natur. Sie benehmen sich wie numerische Variablen außer in den folgenden Fällen, in denen sie sich wie CHAR (nn) Variablen verhalten:
- Wenn sie mit Feldern oder Konstanten des Typs CHAR oder VARCHAR oder Views verglichen werden.
- Wenn sie als Quelle einer MAP oder OVERLAY-Anweisung erscheinen, deren Ziel ein Feld des Typs CHARACTER oder VARCHAR oder ein View ist.

Views

Die von der CASE-Einrichtung der vorliegenden Erfindung verwendeten Daten sind in hierarchischen Strukturen organisiert. Zum Beispiel ist ein Unter-View der Vorgänger eines Felds.

Zur Erleichterung der Dokumentation werden Views von links nach rechts in ausgeschriebener Form wiedergegeben anstelle der grafischen von oben nach unten Methode. Der oberste (am weitesten links befindliche) Knoten wird auch als der "01-Level View" bezeichnet (eine COBOL und PL/1 Konvention).

Die folgende Datenstruktur weist VW_1 als ihren 01-Level View auf. VW_1 besteht aus den Nachfolger-Views CG_VW_11, CG_VW_13 und CG_VW_10, und den Feldern CUSIP_DESCR, CUST_BAL_AMT und CASH_TXN_BAL_AMT. Der Nachnachfolger-View (auch bezeichnet als

Nachfolger-View oder Nachfolger von) CG_VW_13 von VW_1 is aus CG_DATE, SUB_VIEW, CG_BIT aufgebaut. Der Nachfolger SUB_VIEW von CG_VW_13 weist die Felder CGVARCHAR und CG_CHAR, aber keine untergeordneten Views auf.

```
VW-1,

     CG_VW_11

          CG_DATE          CHAR       (08),
          CG_VARCHAR       VARCHAR    (10),
          CG_CHAR          CHAR       (10),
          CG_BIT           CHAR       (01),

CG_VW_13,

          CG_DATE          CHAR       (08),
          SUB_VIEW         (05)             *>tritt 5 x auf<*
            CG_VARCHAR     VARCHAR    (10),
            CG_CHAR        CHAR       (10),
          CG_BIT           CHAR       (01),

     CUSIP_DESCR           CHAR (20),
     CUST_BAL_AMT          DECIMAL (15,2),
     CASH_TXN_BAL_AMT      PIC S9999V99,

     CG_VW_10

          SUB_VIEW       (05),          *>tritt 5-mal auf<*
               CG_VARCHAR      VARCHAR   (10),
               CG_CHAR         CHAR (10),
          INT_RATE             DECIMAL   (7,7),
          CG_BIT               CHAR      (01),
```

Fig. 5 Datenstruktur eines Views

Der obige View, VW_1, erläutert viele der Konzepte der zugrundeliegenden HPS-Datenstrukturen. Jeder der Unter-Views, der zu der obigen Baumstruktur gehört, ist seinerseits eine Baumstruktur, bei der er der 01-Level View wird. Zum Beispiel definiert CG_VW_10 einen Baum mit CG_VW_10 als oberster Knoten, SUB_VIEW, INT_RATE und CG_BIT als seine Blätter und Knoten vom zweiten Level und CG_VARCHR und CG_CHAR als seine Blätter vom dritten Level.

Ein View ist durchgängig innerhalb des Endlagers durch den Namen seines 01-Level Knotens festgelegt. Mit anderen Worten steht VW_1 für die gesamte Sammlung aller Views und Felder in der Figur.

Views und Felder können mehr als einmal innerhalb eines Baums auftreten. Es ist ihnen aber nicht erlaubt, durch Bezugnahme auf sich selbst weder direkt noch durch eine Kette von dazwischen liegenden Nachfolger- oder Vorgänger-Views rekursive Konstrukte zu bilden.

Zum Beispiel ist CG_CHAR ein Feld, und SUB_VIEW ist ein View, die jeweils mehr als einmal innerhalb von VW_1 auftreten.

Es ist anzumerken, daß gemäß Paragraph 1, SUB_VIEW dieselbe Baumstruktur definiert, unabhängig davon, ob er als Nachfolger von CG_VW_10 oder CG_VW_13 auftritt.

```
SUB_VIEW              (05),
     CG_VARCHAR            VARCHAR    (10),
     CG_CHAR               CHAR       (10),
```

Fig. 6 Unter-View Datenstruktur

Das Feld CG_CHAR, das zu dem View SUB_VIEW gehört, der zu dem View CG_VW_10 gehört, kann andere Daten als das Feld CG_CHAR enthalten, das zu SUB_VIEW gehört, der zu CG_VW-13 gehört. Tatsächlich belegen sie unterschiedliche Speicherorte. Weiterhin erscheint CG_CHAR auch anderswo -- es ist auch eine Komponente von CG_VW_11. Wie hält man diese Dinge innerhalb eines Regelprogramms auseinander?

Zuerst schreibt man den vollen qualifizierten Namen der als doppeldeutig bestimmten Gegenstände nieder (die entweder Felder oder Views sein können). Der volle qualifizierte Namen beginnt mit dem Namen der Datengröße auf dem niedrigsten Level (die entweder ein Feld oder ein View sein kann) und arbeitet sich zu der Spitze des Baums unter Benennung jedes Views (nicht Felds) und Trennen jedes der Namen mit dem reservierten Wort OF hoch. (Diese Benennungskonvention ist COBOL entlehnt und unterscheidet sich von derjenigen, die in C und PL/I verwendet wird.) Zum Beispiel haben wir im Fall von CG_CHAR

```
CH_CHAR OF SUB_VIEW OF CG_VW_11 OF VW_1,
CG_CHAR OF SUB_VIEW OF CG_VW_13 of VW_1 und
CG_CHAR OF CG_VW_10 OF VW_1
```

Fig. 7 Volle qualifizierte Namen

Die grundsätzliche Notation ist die folgende:

> X sei ein Feld oder View, das/der einen Vorgänger V1 hat, der einen Vorgänger V2 hat ... usw..
>
> Vn bezeichne den obersten Knoten ...
>
> Dann ist der volle qualifizierte Name von X:
>
> X OF V1 OF V2 ... OF Vn.

Diese vollen qualifizierten Namen sind ausreichend, um zwischen den Fällen von CG_CHAR zu unterscheiden. Tatsächlich enthalten sie redundante Informationen für den Regelcodeerzeuger, um zwischen den Fällen zu unterscheiden. Um die notwendigen und ausreichenden Informationen zu erhalten, kann man die Namen der Views, die den vollen qualifizierten Namen gemeinsam sind, einfach streichen. Natürlich darf man nicht die Namen an der

Basis des Wegs aufwärts zu dem obersten Level View streichen. Wendet man dies auf das obige Beispiel an, ergibt sich:

```
CG_CHAR        OF CH_VW_11
CH_CHAR        OF CG_VW_13 und
CG_CHAR        OF CG_VW_10
```

Fig. 8 Editierte qualifizierte Namen

Dies ist die minimale Information, die der Regelcodeerzeuger benötigt, um zwischen den Fällen von CG_CHAR zu unterscheiden. Weiterhin ist dies auch die minimale Information, die ein Auge braucht, um die Fälle mit einem Blick auf eine Darstellung von VW_1 zu unterscheiden.

Es gibt jedoch einen Typ von Doppeldeutigkeit, der nicht mit vollem qualifizierten Namen gelöst werden kann, falls qualifizierte Teilnamen erlaubt sind. Man betrachte die folgende View-Struktur:

```
VIEW10
   ├── FIELD1,          *>erstes Auftreten von FIELD1<*
   ├── VIEW1,
   │      ├─── FIELD1   *>zweites Auftreten von FIELD1<*
   │      └─── FIELD2,  *>erstes Auftreten von FIELD2<*
   └── VIEW2,
          └─── FIELD2,  *>zweites Auftreten von FIELD2<*
```

Fig. 9 Zweideutige Feldstruktur

VIEW0 enthält FIELD1 als eine zweideutige Bezugnahme, weil die einzige vernünftige Wahl für (teilweise qualifizierte) Namen für das erste Auftreten entweder:

```
FIELD1              oder      FIELD1 OF VIEW 10
```

ist, aber beide Namen auch auf das zweite Auftreten von FIELD1 Bezug nehmen. Im Gegensatz dazu ist zu bemerken, daß man einen deutlichen Unterschied zwischen den Inhalten von

```
FIELD2 OF VIEW1     und       FIELD2 OF VIEW2
```

machen kann.

Tiefindizes

Die Regelsprache unterstützt Views mit Tiefindizes (das OCCURS-Attribut der View-View/HPS-Beziehung). Sie sind die Gegenstücke zu Tabellen in COBOL und Arrays in C oder PL/1.

Angenommen, daß VW_2 ein in dem HPS-Endlager wie folgt definierter View ist:

```
VW_2

     CG_VW_11          (8)       *>tritt 8-mal auf<*
```

```
     CG_DATE          DATE
     CG_VARCHAR       VARCHAR    (10),
     CG_CHAR          CHAR       (10),
     CG_BIT           CHAR       (10),

CH_VW_13              (10),      *>tritt 10-mal auf<*
     CG_DATE          DATE
     CG_SUB_VIEW      (05),      *>tritt 5-mal auf<*
        CG_VARCHAR    VARCHAR         (10),
        CG_CHAR       CHAR            (10),
        CG_BIT        CHAR            (01),

CG_VW_10
   CG_VARCHAR         VARCHAR         (10),
   CG_CHAR            CHAR            (10),
   CG_BIT             CHAR            (01),
```

Fig. 10 View mit Indizes

Bei jedem Auftreten von VW_2 gibt es 8 Fälle von CG_VW_11 (und allen ihm untergeordneten Feldern), 10 Fälle von CG_VW_13 (und allen Feldern und Views darunter) und 5 Fälle von CG_SUB_VIEW, die jedem der 10 Fälle von CG_VW_13 nachgeordnet sind.

Anmerkung: Eine Auftrittshäufigkeitsklausel ist kein Besitz eines Viewelements, sondern der Beziehung (OWNS), die zwischen einem View und einem anderen erzeugt wurde.

Unter Verwendung des Beispiels in Fig. 10 kann auf CG_CHAR Bezug genommen werden wie folgt:

```
1.   CG_CHR OF CG_SUB_VIEW OF CG_VW_13 OF VW_2     (S1)
2.   CG_CHR OF CG_SUB_VIEW OF CG_VW_13 OF VW-2    (S1, S2)
3.   CG_CHR OF CG_SUB_VIEW OF CG_VW_13 OF VW-2    (S1, S2, S3)
```

wobei S1, S2, S3 die Tiefindizes von VW_2, CG_VW_13 bzw. CG_SUB_VIEW bezeichnen. Es ist festzustellen, daß in konventioneller Ausdrucksweise die obige Zahl 1 einem zweidimensionalen Array entspricht, die obige Zahl 2 einem eindimensionalen Array und die obige Zahl 3 einer Einfeldgröße.

Keinem Feld, das in der Regelsprache verwendet wird, ist es erlaubt, mehr als drei (3) Auftrittshäufigkeitsattribute in irgendeinem seiner Qualifikationswege aufzuweisen. Es ist auch anzumerken, daß die Tiefindizes so geschrieben sind, daß der am weitesten links befindliche Index dem obersten View mit mehrfachem Auftreten entlang des Pfads von X zu Vn entspricht.

Ein View kann bis zu 20 Levels tief geschachtelt werden. Mit anderen Worten, falls die Levelnummern wie folgt ansteigen:

```
01   03   05   07   ...
```

ist die höchste unterstützte Levelnummer 39.

Konstanten

Konstanten treten in der Regelsprache in zwei Manifestationen auf:

. Literale und
. Setsymbole.

Sie können repräsentieren:

. Zeichen, wie beispielsweise "New York City", oder
. (relative) ganze Zahlen, so wie 123 oder 1234567, oder
. (relative) Dezimalzahlen, so wie 123.45 oder 1234.50.

Numerische Literale

Es gibt zwei Arten von numerischen Literalen. Ganze Zahlen und Dezimalzahlen. Eine Ganzzahl ist eine Folge von einer oder mehreren Stellen. Eine Dezimalzahl ist eine Folge von einer oder mehreren Stellen gefolgt von einem Dezimalpunkt, dem 0 oder mehr Stellen folgen können. Eine negative Zahl wird ausgedrückt, indem ein Minuszeichen den Stellen voransteht.

Zum Beispiel sind 0 und 123 Ganzzahlen. 123.0, -7734.33 und 123.456 sind gültige Dezimalkonstanten ebenso wie 0.456.

Für diese Literale bestehen abhängig von ihrem Typ (Ganzzahl oder Dezimalzahl) Beschränkungen bezüglich der Anzahl der Stellen, die sie enthalten. Ganzzahlen bestehen aus bis zu 15 Stellen, während Dezimalzahlen bis zu 16 Stellen enthalten (15 + 1 für den Dezimalpunkt). Positiven Zahlen muß kein Pluszeichen voranstehen, aber negativen Größen muß ein Minuszeichen voranstehen.

Zeichenliterale

Ein Zeichenliteral ist ein '(einzelnes Hochkomma), gefolgt von null bis 50 Zeichen (andere als '), gefolgt von einem '. Das folgende sind gültige Zeichenliterale.

```
'Dies ist eine gültige Zeichenkonstante'
'ZYZZY und PLUGH sind magische Worte'
```

Das letzte Beispiel ist eine Null-Reihe. Falls es notwendig ist, ein einzelnes Hochkommazeichen in das Literal selbst einzubringen, kann der übliche Trick der Verwendung von zwei aufeinander folgenden einzelnen Hochkommata angewendet werden. Zum Beispiel sind gleich

```
'Mike' 's computer is broken.'      und

Mike's computer is broken.
```

Es ist anzumerken, daß einzelne Hochkommata verwendet werden **müssen**. Doppelte Anführungsstriche sind **nicht** gültig.

Sets und Symbole

Ein Set ist eine Zusammenstellung von Datengrößen konstanten

Werts. Alle Konstanten sind vom selben Datentyp. Zum Beispiel sind alle CHAR(6) oder alle sind SMALLINT. Aufgrund dieser Eigenschaft können wir auf den Datentyp eines Sets Bezug nehmen. Die Konstanten eines Sets verhalten sich wie Felder und nicht wie Views in folgendem Sinne:

- Sie können nicht in niedrige Levels aufgebrochen werden und
- sie können keine Auftrittshäufigkeitsklausel aufweisen.
- Zusätzlich kann der gesamte Set keine Auftrittshäufig keitsklausel haben.

Beispiel:

```
SET       MONTHSET      SMALLINT,
            JAN         VALUE 1,
            FEB         VALUE 2,
            MAR         VALUE 3,
            APR         VALUE 4,
            MAY         VALUE 5,
            JUN         VALUE 6,
            JUL         VALUE 7,
            AUG         VALUE 8,
            SEP         VALUE 9,
            OCT         VALUE 10,
            NOV         VALUE 11,
            DEC         VALUE 12;
```

Das Beispiel definiert einen MONTHSET genannten Set, der aus einer Sammlung von zwölf Konstanten zusammengesetzt ist, die die zwölf Monate eines Jahres wiedergeben. Ihre Symbole sind JAN, FEB, ... DEC, und ihre zugeordneten Werte sind ganzzahlige Literale 1, 2, ..., 12.

MONTHSET ist vom Typ SMALLINT, was bedeutet, daß das Format der Werte SMALLINT ist. Es kann keinen VALUE 3.21 und keinen VALUE "NOT AN SMALLINT" und keinen VALUE 1234565 geben. (Das letzte Beispiel ist ungültig, weil ein SMALLINT nicht die Zahl 32767 überschreiten kann.)

Dieses Beispiel wird für weitere Erläuerungen des Setkonzepts verwendet werden.

Jede der Konstanten eines Sets besitzt eine Bezeichnung, genannt ein SYMBOL, und einen Wert. Das SYMBOL ist ein Hilfsmittel zur Bezugnahme auf den zugrundeliegenden Wert. Ein Setsymbol kann in derselben Weise verwendet werden wie ein Feld. Der grundsätzliche Unterschied ist, daß der Wert eines Setsymbols konstant ist und niemals durch eine Regelanweisung geändert werden kann, während der Wert eines Felds geändert werden kann; z. B. durch Löschen oder indem das Feld zum Ziel einer MAP- oder OVERLAY-Anweisung gemacht wird.

Angenommen SYMXXX ist ein Symbol, das zu einem Set SETYYY

gehört. Dann kann auf das Symbol auf eine der folgenden Weisen Bezug genommen werden:

```
SYMXXX

SYMXXX IN SETYYY
```

Das IN-Schlüsselwort wurde anstelle des OF-Schlüsselworts gewählt, um eine Überfrachtung der Bedeutung des letzteren bei zu vielen unterschiedlichen Verwendungen zu vermeiden. SYMXX muß durch seinen Set qualifiziert werden, falls SYMXX in der Regel verwendet wird, entweder als Symbol eines anderen Sets oder als ein Feldname oder als ein Viewname.

Rückbezugnehmend auf das MONTHSET-Beispiel:

```
MAR
MAR IN MONTHSET
3
```

beachte, daß alle exakt dieselbe Bedeutung haben, nämlich die Zahl 3.

```
MAP  FEB IN MONTHSET          TO XXXVAR OF YYYVIEW
```

hat den Effekt des Verschiebens des Werts 2 in das Feld XXXVAR, das hoffentlich irgendwo innerhalb der Regel oder seines Verbindungspfeils richtig als numerisches Unterfeld des View YYYVIEW erklärt wurde.

Innerhalb eines Sets kann es nicht zwei oder mehr Symbole mit demselben Symbolnamen geben. Es ist aber anzumerken, daß ein Set verschiedene Symbole mit einem und demselben Wert besitzen kann!

Andere Bezeichnungen

Neben Datengrößen werden die folgenden Elemente in einer Regel verwendet:

- Regelname
- Komponentenname
- Fenstername
- Setname

Sie werden im Detail während der Diskussion der ausführbaren Anweisungen beschrieben werden, wo sie auftreten.

Lokale Erklärungen

Gelegentlich ist es bequem, lokal Variablen in einem Regelprogramm zu haben. Eine Variable, die als indizierende Variable eines indizierten DO verwendet wird, ist ein gutes Beispiel für eine solche Variable. Die DCL ...ENDDCL-Anweisung ermöglicht die Erklärung von lokalen Variablen.

Syntax

Die DCL ... ENDDCL-Anweisung hat die Form:

```
DCL
     Erklärung; [Erklärung; ...]
ENDDCL,
```

wobei die Erklärung entweder

```
     [Bezeichnung [ (S) ]
[,   Bezeichnung [ (S) ], Erklärungs_Typ
```

oder

```
EXTERNe Bezeichnung
     [, Bezeichnung, ...]  SET
```

ist, wobei "(S)" ein Tiefindex (ganzzahlige Kostante in Klammern) ist.

Der Erklärungstyp ist einer der folgenden:

```
SMALLINT
INTEGER
CHAR
LIKE bereits_erklärte_Datengröße
```

Anmerkung:
Erklärungen des Typs EXTERN können nicht tiefindiziert werden. "Bezeichnung" muß mit einem alphabetischen Zeichen beginnen. Die folgenden Zeichen sind entweder alphabetisch, numerisch oder Unterstriche. Bereits_erklärte_Datengröße bezieht sich auf eine Größe, die der Regel bereits durch eine Vorbereitung von dem HPS-Endlager oder durch eine zuvorige lokale Erklärung bekannt ist.

Ein Beispiel folgt:

```
DCL
     I, J, K, L, M, NM        INTEGER;
     SSN                      CHAR(11);
     TEMP_ID                  CHAR(#@);
     TEMP_VIEW                LIKE RTAXCMPI(5);
ENDDCL
```

Fig. 10 Beispiel einer DCL ... ENDDCL-Anweisung

Es ist anzumerken, daß es **nicht** möglich ist, einen View direkt innerhalb einer DCL ... ENDDCL-Anweisung zu definieren. Es ist möglich, so etwas indirekt mit der LIKE-Klausel zu tun. In dem obigen Beispiel ist TEMP_VIEW ein zeitweiliger lokaler View mit nahezu derselben Struktur wie RTAXCMPI. Der einzige Unterschied ist, daß der 01-Levelname TEMP_VIEW anstelle von RTAXCMPI ist und daß TEMP_VIEW 5-mal auftritt.

ANHANG D: Ausführbare Anweisungen der Regelsprache

Zuordnung von Werten

Drei Arten von Anweisungen ändern die Werte von Daten, die in Feldern und Views enthalten sind: MAP, CLEAR und OVERLAY. Die Syntax und Semantik jeder der Anweisungen wird in der Folge beschrieben werden.

MAP-ANWEISUNG

Syntax

Die Syntax der MAP-Anweisung ist:

```
MAP Daten_Größe TO Variable,
```

wobei Daten_Größe eine Konstante, ein Feldname oder ein Viewname und Variable entweder ein Feldname oder ein View-Name ist. Man erinnere, daß die Feld- oder Viewnamen unzweideutig qualifiziert sein müssen, und daß diese Namen Tiefindizes aufweisen können.

Einige Beispiele folgen:

```
MAP  'Syntax' TO CG_CHAR OF CG_VW_11

MAP CG_CHAR OF CG_VW_11 TO CG_CHAR OF SUB_VIEW OF
CG_VW_13 (3)

MAP 23400.00 TO CUST_BAL_AMT
```

Fig. 1 Beispiel von MAP-Anweisungen

Semantik

Das Resultat der Abbildung z. B. von A auf B kann in Abhängigkeit von den Datentypen von A und B stark variieren. Wie jene Zuordnungen arbeiten, kann man durch Studieren der folgenden Matrize sehen, die gemäß dem folgenden Prinzip aufgebaut ist:

Die mögliche Quellen (A) für eine MAP-Anweisung bildet die Zeilen der Matrix, und die möglichen Bestimmungen (B) bilden ihre Spalten. Die Richtigkeit der Syntax bzw. das Ergebnis der MAP-Operation wird am Schnittpunkt der Reihe und Zeile gegeben. Die Zahlen in Klammern beziehen sich auf die numerierten Paragraphen, die der Matrize unmittelbar folgen. VW(5) und VW(8) sind zwei Views -- beide mit mehrfachen Auftritten, aber einer mit weniger (5) und der andere mit mehr (8).

Quelle (A)	(B) Bestimmung				
	View	Feld	Konstante	VW(5)	VW(8)
View	OK (1)	FEHLER	FEHLER	WARNUNG (3)	WARNUNG (3)
Feld	FEHLER	(4)	FEHLER	FEHLER	FEHLER
Numer. Literal	FEHLER	(6)	FEHLER	FEHLER	FEHLER
Zeichen-literal	FEHLER	(5)	FEHLER	FEHLER	FEHLER
Set	FEHLER (2)	FEHLER	FEHLER	FEHLER	FEHLER
Symbol	FEHLER	(6)	FEHLER	FEHLER	FEHLER
VW(5)	WARNUNG	FEHLER	FEHLER	OK	WARNUNG
VW(8)	WARNUNG (3)	FEHLER	FEHLER	WARNUNG (3)	OK

Fig. 2 Die MAP-Operation

(1) Das Abbilden eines Views auf einem View bedeutet das Abbilden aller Unter-Views und/oder Felder mit entsprechenden Namen, die dem Quell-View und dem Bestimmungs-View untergeordnet sind. Zum Beispiel angenommen, man hat:

```
VIEW_1                          VIEW_2
     ABC                             OPQ
     DEF tritt auf (50)              ABC
     OPQ                             DEF tritt auf (9)
     XXX                             BBB
     BBB                             ZZZ
     XXY
     XYZ
```

wobei die Spezifikationen der Unter-Views und der Felder von VIEW_1 und VIEW_2 **unerheblich** sind. Dann hat MAP VIEW_1 TO VIEW_2 den folgenden Effekt:

```
ABC OF VIEW_1          wird verschoben nach  ABC OF VIEW_2
OPQ OF VIEW_1          wird verschoben nach  OPQ OF VIEW_2
BBB OF VIEW_1          wird verschoben nach  BBB OF VIEW_2
DEF OF VIEW_1 (1)      wird verschoben nach  DEF OF VIEW_2 (1)
DEF OF VIEW_1 (2)      wird verschoben nach  DEF OF VIEW_2 (2)
DEF OF VIEW_1 (5)      wird verschoben nach  DEF OF VIEW_2 (5)
```

Anmerkungen:
Nichts wird von xxx, xxy, xyz irgendwo hin abgebildet, weil VIEW_2 **keine** direkten Nachfolger mit diesen Namen unter sich hat. Nichts wird auf zzz abgebildet, weil VIEW_1 keinen Nachfolger mit diesem Namen direkt unter sich hat.

Angenommen, daß xxx und zzz ihrerseits Views sind, die wie folgt aufgebaut sind:

```
xxx              zzz
FLD1             FLD1
     FLD2             FLD3
     FLD3             FLD5
```

dann wird

FLD1 OF xxx OF VIEW_1 nicht nach FLD1 OF zzz OF VIEW_2 verschoben und

FLD3 OF xxx OF VIEW_1 nicht nach FLD3 OF zzz OF VIEW_2 verschoben.

Der Grund ist, daß nur die Nachfolger direkt unter der Quelle (= VIEW_1) und dem Ziel (= VIEW_2) der MAP-Anweisung in Frage kommende Kandidaten für übereinstimmende Namen sind. Das heißt, der Codeerzeuger schaut nur ein Level abwärts, wenn er Vergleiche der View-Strukturen durchführt.

(2) Ein Set ist weder eine Konstante noch eine Variable, und absolut nichts kann von einem oder auf einen Set abgebildet werden.

(3) Angenommen, daß A nn-mal auftritt und B mm-mal auftritt und daß min die kleinere Zahl von mm und nn bezeichnet, dann bedeutet:

```
MAP A TO B
```

genau dasselbe wie das folgende Regelfragment:

```
DO FROM 1 TO min INDEX IX
          MAP A(IX) To B(IX)

ENDDO
```

Falls entweder A mehrmals auftritt und B nicht oder A einmal auftritt und B mehrmals, dann ist MAP A TO B entweder äquivalent zu

```
MAP A (1) TO B
```

oder

```
MAP A TO B(1)
```

(4) Siehe die Feld-zu-Feld Abbildung Fig. 2. Ein Quell-Literal des Typs INTEGER oder DECIMAL benimmt sich genau so wie eine Variabel dieses Typs.

(5) Siehe die Feld-zu-Feld Abbildung in Fig. 2. Ein Quell-Literal der Form 'ABCDEFGH' benimmt sich genau so wie eine Variable des Typs CHAR(8).

(6) Siehe die Feld-zu-Feld Abbildung in Fig. 2. Eine Quelle, die ein Symbol ist, das zu einem Set eines gegebenen Typs gehört, benimmt sich genau so wie eine Variabel desselben Typs.

Abbilden von VARCHAR-Feldern

Im folgenden ist beschrieben, wie MAP-Anweisungen mit Variablen des Typs VARCHAR arbeiten.

Der Hauptunterschied zwischen Variablen des Typs VARCHAR (nn) und jenen des Typs CHAR (nn) beruht auf der Tatsache, daß VARCHAR (nn)-Variablen ein zugeordnetes Längenfeld haben, genannt xxx_LEN für eine VARCHAR-Variable xxx. xxx_LEN enthält die "aktuelle Länge" von xxx, während nn die "maximale Länge" von xxx ist.

Angenommen, daß B eine Variable des Typs VARCHAR mit einer Länge B_LEN ist, und daß A eine Variable, ein Literal, ein Symbol des Typs VARCHAR oder vom Typ "CHAR" ist. Dann wird B_LEN von der Länge von A in einer MAP-Anweisung A TO B wie folgt festgelegt.

Falls die Länge von A <= die maximale Länge von B ist, dann:

B_LEN = Länge von A
Inhalte von B = Inhalte von A aufgefüllt mit Leerzeichen nach rechts

Falls die Länge von A > die maximale Länge von B ist, dann:

B_LEN = maximale Länge von B
Inhalte von B = die ersten "maximale Länge von B" Zeichen von A

Beispiele der Verwendung von VARCHAR-Variablen
Angenommen, daß wie folgt:

CHAR_VAR1	vom Typ CHAR (10)
CHAR_VAR2	vom Typ CHAR (20)
VARCH_VAR1	vom Typ VARCHAR (15)
VARCH_VAR2	vom Typ VARCHAR (20)

Weiter angenommen, daß man eine Regel hat, die aus den folgenden Abbildungsanweisungen besteht:

```
1.   MAP 'ABC   '                        TO CHAR_VAR_1
2.   MAP '**MEINE LÄNGE IST 20**'        TO CHAR_VAR_2
3.   MAP 'ABC   '                        TO VARCH_VAR_1
4.   MAP CHAR_VAR_1                      TO VARCH_VAR_1
5.   MAP CHAR_VAR_2                      TO VARCH_VAR_1
6.   MAP CHAR_VAR_2                      TO VARCH_VAR_2
7.   MAP VARCH_VAR_2                     TO VARCH_VAR_2
8.   MAP VARCH_VAR_1                     TO VARCH_VAR_2
```

Dann wird das folgende Ergebnis bei den VARCHAR-Variablen auftreten

#1 und 2 Nichts wurde bislang VARCH_VAR_1_LEN und VARCH_VAR_2_LEN zugeordnet,

#3 VARCH_VAR_1_LEN = 5 = Länge von 'ABC ',

#4 VARCH_VAR_1_LEN = 10 = Länge von CHAR_VAR_1,

#5 VARCH_VAR_1_LEN = 15 = Länge von VARCH_VAR_1, der Inhalt von VARCH_VAR_1 wird sein '**MEINE LÄNGE I',

#6 VARCH_VAR_2_LEN = 20 = Länge von VARCH_VAR_2, der Inhalt von VARCH_VAR_2 wird sein '**MEINE LÄNGE IST 20**'

#7 VARCH_VAR_2_LEN = 20 = Länge von VARCH_VAR_2, der Inhalt von VARCH_VAR_2 wird sein '**MEINE LÄNGE IST 20**'

#8 VARCH_VAR_2_LEN = 15 = Länge von VARCH_VAR_1, der Inhalt von VARCH_VAR_2 werden sein '**MEINE LÄNGE I'

CLEAR-Anweisung

Die CLEAR-Anweisung setzt numerische Felder auf Null und Zeichenfelder auf leer.

Syntax

Die CLEAR-Anweisung hat die Form:

```
CLEAR Variable,
```

wobei "Variable" der volle oder teilweise qualifizierte Name entweder eines Views oder eines Felds ist. Falls die gelöscht werdende Variable Tiefindizes aufweist, das heißt, daß irgendwelche ihrer Views oder Sub-Views mehrfach auftreten, dann können auch Tiefindizes erscheinen. Zum Beispiel setzt

```
CLEAR VW_1
```

alle numerischen Felder von VW_1 auf Null und leert die

Zeichenfelder. Man erinnere, daß numerische Felder INTEGER (n), DECIMAL (p, q) und Felder, die mit PIC-Elementen S, 9 und V beschrieben sind, einschließen. Zeichenfelder schließen CHAR (n) und VARCHAR (n) ein.

Semantik

Ein Feld des zugrundeliegenden Typs "Zeichen" (nicht einschließend PICs) wird auf Leerzeichen gesetzt. Ein Feld des zugrundeliegenden Typs "numerisch" (einschließlich Dezimalzahlen und PICs) wird auf null gesetzt.

Ein View, sagen wir V, wird wie folgt behandelt:

(1) Angenommen, daß V Felder besitzt, aber keinen View, dann wird jedes dieser Felder auf Leerzeichen oder null gesetzt, abhängig davon, ob sein zugrundeliegender Typ "Zeichen" oder "numerisch" ist. Falls V ein View ist, der mit Tiefindizes versehen ist (die entsprechende OCCURS-Klausel könnte möglicherweise auf einem höheren Level als demjenigen von V festgelegt sein), dann bedeutet das Löschen von V dasselbe wie das Löschen von jedem V(S1) oder V(S1, S2) oder V(S1, S2, S3). Dabei bezeichnen S1, S2 und S3 die Tiefindizes von V und hängen von der Anzahl der darüber stehenden OCCURS ab; das Löschen schließt den Level von V ein, unabhängig davon, ob V einmal, zweimal oder dreimal tiefindiziert ist.

(2) Falls V einen oder mehrere Views V1, V2, ... als Nachfolger hat, dann behandele V wie folgt: Lösche die Felder, die die Nachfolger von V sind, genau wie es oben dargelegt wurde. Dann Prüfe jedes V1, V2, um festzustellen, ob es auch Felder als Nachfolger enthält, und falls ja, behandele sie gemäß oben (1). Anderenfalls überprüfe jeden Unter-View.

Das Netz, das aus dem obigen Algorithmus resultiert, ist wie folgt. Jeder View wird letztlich in einen Set von Feldern aufgeteilt. Jedes dieser Felder wird zu Leerzeichen oder auf null gesetzt, abhängig davon, ob sein zugrundeliegender Typ "Zeichen" oder "numerisch" ist.

OVERLAY-Anweisung

Diese Anweisung wird verwendet, um eine Struktur in einem Speicher durch den Inhalt einer anderen zu ersetzen ("überdecken").

Syntax

Die OVERLAY-Anweisung hat die Form

```
OVERLAY Datengröße TO Variable,
```

wobei Datengröße eine Konstante, ein Feld oder ein View ist, und Variable entweder der Name eines Views oder eines Felds ist.

Semantik

Wie in dem Fall der MAP-Anweisung ist die Auswirkung von OVERLAY komplex und hängt von den Typen der Quell- und Bestimmungsvariablen ab.

Quelle (A)	Bestimmung				
	View	Feld	Konstante	VW(5)	VW(8)
View	OK (1)	(4)	FEHLER	WARNUNG (3)	WARNUNG (3)
Feld	(4)	FEHLER	FEHLER	WARNUNG (3)	WARNUNG (3)
Numer. Literal	FEHLER	FEHLER	FEHLER	FEHLER	FEHLER
Zeichen-literal	(5)	FEHLER	FEHLER	FEHLER	FEHLER
Set	FEHLER (2)	FEHLER	FEHLER	FEHLER	FEHLER
Symbol	(10)	FEHLER	FEHLER	FEHLER	FEHLER
VW(5)	WARNUNG (3)	WARNUNG (3)	FEHLER	OK	WARNUNG (3)
VW(8)	WARNUNG (3)	WARNUNG (3)	FEHLER	WARNUNG (3)	OK

Fig. 3 Die OVERLAY-Operation

(1a) Die OVERLAY-Anweisung arbeitet auf eine fundamental andere Weise als die MAP-Anweisung. Falls die Quell- und Bestimmungs(D)-Variablen beides Views sind, dann hat OVERLAY S TO D denselben Effekt wie eine gerade COBOL-Verschiebung. Sowohl S als auch D werden behandelt, als seien sie vom Typ Zeichen (Character). nn sei die kleinere Länge von Länge(S) und Länge(D), dann wird die obige Anweisung einfach die am weitesten links stehenden n Zeichen vom S in die am weitesten links stehenden nn Zeichen von D überführen, wobei die überschüssigen Bytes von D mit Leerzeichen aufgefüllt werden, falls Länge(D) größer ist als Länge(S).

(1b) Falls A ein View und B ein Feld ist, dann ist OVERLAY A TO B OK, solange wie B vom Typ "Zeichen" ist (was bei einem PIC-Feld der Fall ist).

(1c) Entweder A oder B oder beide müssen ein View sein. Eine Konstante kann mit überhaupt nichts überdeckt werden.

(2) Ein Set ist weder eine Konstante noch eine Variable, und nichts kann von einem Set überdeckt werden oder einen Set überdecken.

(3) Angenommen, daß A nn-mal auftritt und B mm-mal auftritt und daß min den kleineren Wert von mm und nn bezeichnet, dann bedeutet

```
OVERLAY A TO B
```

genau dasselbe wie das folgende Regelfragment:

```
DO FROM 1 TO min INDEX IX
    OVERLAY A(IX) TO B(IX)
ENDO
```

Falls entweder A mehrfach auftritt und B nicht oder A einmal auftritt und B mehrfach, dann ist OVERLAY A TO B entweder äquivalent zu

```
OVERLAY A(1)    TO B
```

oder

```
OVERLAY A       TO B(1)
```

(4) Es ist erlaubt, ein Feld mit einem View oder umgekehrt zu überdecken, so lange das Feld vom Typ "Zeichen" ist. Der Grund ist, daß Views selbst als vom Typ "Zeichen" angenommen werden (COBOL-Konvention).

(5) Es ist erlaubt, einen View mit einer Konstanten (Literal oder Symbol) zu überdecken, solange die Konstante vom Typ "Zeichen" ist.

ANHANG E: Steuerfluß bei einer Regel

Die Regelsprache verwendet die üblichen Steuerflußkonstrukte: IF und IF ... ELSE für Bedingungsausführungen, CASEOF für die Auswahl von einer von mehreren Alternativen und DO ... WHILE für die Steuerung von wiederholten Schleifen.

Bedingungen

Eine Bedingung ist eine Mischung von Datengrößen, Verhältnisoperatoren, Boole'schen Operatoren und Klammern. Eine Datengröße umfaßt Zeichen-, ganzzahlige und dezimale Konstanten, Felder und Views. Die Verhältnisoperatoren und Boole'schen Operatoren sind:

Operator	Bedeutung
=	gleich
<>	nicht gleich
<	weniger als
<=	weniger als oder gleich
>	größer als
>=	größer als oder gleich
INSET	ist enthalten in
AND	Konjunktion
OR	Adjunktion
NOT	Verneinung

Fig. 1 Verhältnisoperatoren und Boole'sche Operatoren

Die Bedingung ist entweder eine einmalige Verhältnisbedingung (im folgenden eine "einfache Bedingung" genannt) oder zwei oder mehrere einfache Bedingungen, die durch die Boole'schen Operatoren verbunden sind. Eine einfache Bedingung hat die Form

Datengröße Verhältnisoperator Datengröße

Einige Beispiele sind:

```
CUST_BAL_AMT < 1000
GROSS_PAY OF RTAXCMPI >= FICA_CUTOFF
CUSTOMER_ID <> '1134-4'
12 INSET MONTH_OF_YEARS
```

Die Regelsprache führt eine Datentypüberprüfung durch; Vergleiche können nur zwischen Datengrößen vom gleichen Typ durchgeführt werden -- Zahlen mit Zahlen und Zeichen mit Zeichen. Alle diese Vergleiche der Größe sind bei numerischen Daten verfügbar, aber nur Tests auf Gleichheit oder Ungleichheit sind bei Zeichen-Daten erlaubt.

Jede einfache Bedingung resultiert in einen Wert von entweder

Sind die folgenden Ausdrücke gültig:

```
JAN IN MONTHSET INSET NAMESET        *>Ergebnisse=FALSE<*
AL  INSET NAMESET                    *>Ergebnisse=TRUE<*
AL <= JAN IN MONTHSET                *>Ergebnisse=FALSE<*
AL <= JAN IN NAMESET                 *>Ergebnisse=TRUE<*
AL <= AL IN NAMESET                  *>Ergebnisse=TRUE<*
AL <= AL NAMESET                     *>Ergebnisse=FALSE<*

SHRTINT INSET NAMESET
LONGINT INSET MONTHSET
```

Natürlich kann zum Zeitpunkt der Codeerzeugung nichts über die Ergebnisse der beiden letzten Ausdrücke gesagt werden, weil sie Variablen involvieren. Es mag den Leser überrascht haben, daß:

JAN IN NAMESET INSET MONTHSET das Ergebnis FALSCH ergibt.

Ein Vergleich von gleichen Typen wird streng durchgeführt. Vergleiche zwischen numerischen Daten und nicht-numerischen sind immer Fehler. Die folgende Tabelle gibt Bedingungen wieder, die beim Vergleich von zwei Zeichentypen oder zwei numerischen Typen auftreten können. Die Bedeutung der Anmerkung (1) variiert jedoch gemäß den verglichenen Typen.

Erste(s)	Zweite(s)		
	Literal	Setsymbol	Variable
Literal	Fehler	Warnung	(1)
Setsymbol	Warnung	Warnung	(1)
Variable	(1)	(1)	OK

Fig. 3 Fehlerbedingungen beim Vergleich von Zeichen und numerischen Größen

(1) Der Vergleich einer Zeichenkonstante mit einer Zeichenvariablen ist erlaubt, solange die Länge der Konstanten kleiner oder gleich der maximalen Länge der Variablen ist. Anderenfalls wird ein Syntaxfehler erzeugt.

Es ist ein Fehler, ein großes (im absoluten Wert) numerische Literal mit einer Variablen zu vergleichen, die "zu klein" ist, um einen so großen Wert aufzunehmen. Zum Beispiel kann ein SMALLINT-Feld Werte von -32,768 bis +32,767 aufnehmen. SMALL_INT sei ein SMALLINT-Feld. Der folgende Vergleich wird vom Regelcodeerzeuger nicht erlaubt:

```
SMALL_INT > 1000000
```

Auch subtilere Fehler werden aufgezeigt. POS-NUMBER sei eine Dezimalzahl, die von dem Bild "PIC 9999/V99" in dem HPS-Endlager

beschrieben wird. Dann ist die nächste Beispielbedingung auch nicht erlaubt:

```
POS_NUMBER < 0.
```

Dies ist ein Fehler, weil POS_NUMBER kein führendes Vorzeichenbildelement S hat.

Anmerkung:

Angenommen bei dem Vergleich von Zeichentypgrößen sei X = 'ABC' und Y = 'ABC' und Z = 'ABC ', dann sind X, Y und Z sämtlich äquivalent, soweit eine Überprüfung auf Gleichheit betroffen ist.

Mit anderen Worten ergeben die komplexen Bedingungen X = Y und Y = Z und X = Z den Wert TRUE. Anzumerken ist weiterhin, daß die zugehörige ASCII-Sequenz und nicht die zugehörige EBCDIC-Sequenz bei einem relativen Vergleich zählt. Bei der oben gegebenen Definition ergibt die Bedingung

```
X >= '1BC'
```

das Ergebnis TRUE, weil der ASCII-Wert von '1' 49 ist, während der ASCII-Wert von 'A' 65 ist.

IF-Anweisung

IF ..., IF ... ELSE (Bedingte Ausführung)

Syntax

Die IF-Anweisung hat die grundsätzliche Form

```
IF Bedingung
     Anweisung ...

[ELSE
     Anweisung ...]

ENDIF
```

wobei "Bedingung" ein logischer Ausdruck ist, der entweder TRUE oder FALSE sein kann. "Anweisung" gibt eine Regelsprachenanweisung wieder und "..." bedeutet, daß die direkt vorangehende Größe wiederholt werden kann, und [] impliziert, daß was innerhalb [und] enthalten ist, optional ist. Ein THEN ist in einer IF-Anweisung nicht erlaubt.

CASEOF-Anweisung

CASE (Auswahl von einer oder mehreren Alternativen)

Die CASEOF-Anweisung wählt auf der Basis des Werts eines Zeichenfelds oder eines numerischen Felds einen von mehreren

TRUE oder FALSE. Zwei dieser Werte können mit den Boole'schen Operatoren AND und OR kombiniert werden, und der Sinn von einem von ihnen kann mit einem NOT umgekehrt werden.

Zum Beispiel ist

```
CUSTOMER_ID <> '1134-4' AND GROSS_PAY >= FICA_LIMIT
```

TRUE, falls sowohl CUSTOMER_ID nicht gleich ist mit 1123-4' und GROSS_PAY größer oder gleich FICA_LIMIT ist; anderenfalls ist die Bedingung FALSE. Es wird angenommen, daß CUSTOMER_ID entweder CHAR, VARCHAR oder ein View ist und daß GROSS_PAY und FICA_LIMIT INTEGER, DECIMAL oder mit numerischen PIC-Elementen beschrieben sind. Weiterhin müssen, damit diese Bedingung Sinn macht, die Beziehungsoperatoren <> und >= angewendet werden, bevor die beiden resultierenden Werte mit dem AND zusammengefügt werden. Dies schließt ein, daß alle Operatoren eine Hierarchie ihres Vorrangs haben. Die übliche Hierarchie wird verwendet und ist in der folgenden Tabelle wiedergegeben.

Operator	Vorrang
INSET	höchster
=, <>, <, <=, >, >=	
NOT	
AND	
OR	niedrigster

Fig. 2 Vorrang - Beziehungsoperatoren und Boole'sche Operatoren

Das Konzept des Vorrangs der Operatoren ist von der täglichen Verwendung arithmetischer Berechnungen gut bekannt, wobei Multiplikation und Division vor Addition und Subtraktion ausgeführt werden. In derselben Weise wird ein Test auf Ungleichheit ausgeführt, bevor zwei Bedingungen durch ein AND verbunden werden, was seinerseits getan wird, bevor irgendwelche Bedingungen durch ein OR verbunden werden.

Klammern können verwendet werden, um den Vorrang der Operatoren zu überstimmen. Zum Beispiel ist

NOT X = Y OR C > D äquivalent mit (NOT X+Y) OR (C>D), was seinerseits äquivalent ist mit X <> Y OR C > D.

Jedoch ist

NOT (X = Y OR C > D) äquivalent mit X <> Y AND C <= D

(weil es in der Regel TRUE ist, daß NOT (X OR Y) dasselbe ist wie (NOT X) AND (NOT Y).

Die Beseitigung des NOT zeigt, daß die zwei Bedingungen recht unterschiedlich sind. So lange wie die Vergleiche zulässig sind, können die Bedingungen willkürlich komplex werden. Zum Beispiel

ist

```
((PAY > LIMIT OR BONUS >= MAX) AND ID <>'112A-3x') OR
ID = ' '
```

syntaktisch korrekt.

INSET-Anweisung

Ein Set kann daraufhin überprüft werden, ob eine gegebene Variable oder Konstante als Wert innerhalb des Sets auftritt.

X sei eine Datengröße, deren Datentyp mit dem Datentyp des Sets kompatibel ist (selbst wenn sie nicht notwendigerweise mit diesem identisch ist).

Dann ist

```
X INSET SET_NAME
```

eine Bedingung, die TRUE oder FALSE in Abhängigkeit davon ergibt, ob oder ob nicht mindestens einer der Werte in SET_NAME den Wert von X trifft.

Es ist nicht erlaubt zu prüfen, ob eine numerische Datengröße innerhalb eines Sets vom Typ Zeichen enthalten ist oder umgekehrt. Es ist anzumerken, daß PIC-Felder und/oder Setsymbole sowohl mit Zeichen- als auch numerischen Datengrößen verglichen werden können. Es ist auch anzumerken, daß bei einem View für Vergleichszwecke angenommen wird, daß er sich wie eine Zeichenvariable verhält.

Unter Annahme der folgenden Erklärungen:

```
CHRVAR1          CHAR (10),
CHRVAR2          CHAR (6),
SHRTINT          SMALLINT;
LONGINT          INTEGER;

SET  MONTHSET          SMALLINT
     JAN                 VALUE 1,
     FEB                 VALUE 2,
     - - - - - - - - - - - - - -
     - - - - - - - - - - - - - -
     - - - - - - - - - - - - - -
     DEC                 VALUE 12;

SET  NAMESET    INTEGER
     AL               VALUE 4
     JAN              VALUE 12345
     JAMES            VALUE 1234567,
     JIM              VALUE 12345677,
     JON              VALUE 123456788
     BOB              VALUE 123456789;
```

Alternativen Ausführungswegen aus. Sie ist in ihrer Bedeutung mehreren geschachtelten IF ... ELSE-Anweisungen äquivalent.

Syntax

Die CASEOF-Anweisung hat die folgende Form:

```
CASEOF Variable

CASE Literal [Literal] ...
     [Anweisung ...]

[CASE Literal [Literal] ...
     [Anweisung ...]] ...

[CASE OTHER
     [Anweisung ...]]

ENDCASE
```

Die hier verwendete Notation, um die Syntax auszudrücken, ist komplexer als diejenige, die für die IF-Anweisung verwendet wurde. Eckige Klammern ([und]) schließen Dinge ein, die weggelassen werden können. Punkte (...) deuten an, daß die direkt voranstehende Größe wiederholt werden kann. Das Zusammenfassen dieser beiden Konventionen bedeutet, daß [Anweisung ...] äquivalent ist mit keine Anweisungen (alles innerhalb [und] ist optional) oder einer Anweisung, zwei Anweisungen (die einer Anweisung folgenden ... bedeuten, daß sie wiederholt werden kann), drei Anweisungen usw..

Semantik

Die CASEOF-Anweisung ist ein kürzerer Weg, um denselben Steuerfluß wie unter Verwendung verschachtelter IF ... ELSE-Anweisungen auszudrücken. Die Variable wird mit der Liste der Literale verglichen, die dem ersten reservierten CASE-Wort folgen. Falls sie einem von diesen gleicht, dann werden die Anweisungen, die dem ersten CASE folgen, ausgeführt bis zu dem aber nicht einschließlich des zweiten CASE; der Steuerfluß geht dann zu den Anweisungen über, die dem ENDCASE folgen.

Falls die Variable keinem der Literale gleicht, die der ersten CASE-Klausel folgen, dann wird sie mit den Literalen verglichen, die dem zweiten CASE folgen. Falls die Variable einem von diesen gleicht, dann werden die Anweisungen, die der zweiten CASE-Klausel folgen, bis zu dem aber nicht einschließlich des dritten CASE ausgeführt; der Fluß geht dann zu den dem ENDCASE folgenden Anweisungen über.

Dieser Prozeß wird wiederholt, bis alle CASE-Klauseln verbraucht sind. Die dem optionalen CASE OTHER folgenden Anweisungen werden nur ausgeführt, falls die Variable bei der CASEOF-Klausel keinem der Literale gleicht, die bei den CASE-Klauseln aufgeführt sind.

Die Literale, die bei den verschiedenen CASE-Klauseln auftreten,

dürfen nur einmal auftreten. Ein Literal, das in einer CASE-Klausel auftritt, kann nicht in einer anderen wiederholt werden.

Im folgenden ist ein Skelettbeispiel der Verwendung eines CASEOF-Konstrukts gezeigt zusammen mit der semantisch identischen Übersetzung in einen Satz von verschachtelten IF-Anweisungen.

```
                    CASE OF TRANS_CODE

                    CASE 'A'   'C'

          Anweisung 1
          Anweisung 2

                    CASE 'M'   'R'   'D'

          Anweisung 3

                    CASE 'X'

          Anweisung 4
          Anweisung 5

                    CASE OTHER

          Anweisung 6

                    ENDCASE
```

Die dem Obigen äquivalente IF-Anweisung folgt:

```
     IF TRANS_CODE = 'A' OR TRANS_CODE = 'C'

          Anweisung 1
          Anweisung 2

     ELSE

          IF TRANS_CODE = 'M' OR TRANS_CODE = 'R'
          OR TRANS_CODE = 'U'

               Anweisung 3

          ELSE

               IF TRANS_CODE = 'X'

                    Anweisung 4
                    Anweisung 5

               ELSE

                    Anweisung 6

               ENDIF
```

```
        ENDIF

    ENDIF
```

Zu bemerken ist, daß die CASEOF-Anweisung implizit Überprüfungen auf die Gleichheit zwischen einer Variablen (Feld oder View) und einer Konstanten verwendet. Die bei der IF-Anweisung durchgeführten Typenüberprüfung wird auch hier durchgeführt. Konstanten, die in den CASE-Klauseln auftreten, müssen denselben Typ wie die Variable haben, die bei der CASEOF-Klausel auftreten.

DO-Anweisung

Das DO ... ENDDO-Konstrukt stellt eine Steuerung von wiederholten Schleifen bereit, und es gibt zwei Varianten -- eine mit einem expliziten Schleifenzählmechanismus und eine ohne.

Syntax

Die Form der DO ... ENDDO-Schleifenkontrollstruktur ist wie folgt:

```
        DO

    [Anweisung ...]

        WHILE Bedingung

    [Anweisung ...]

        ENDDO
```

Die Form mit dem expliziten Schleifenzählmechanismus ist:

```
    DO [FROM ganzzahlige Datengröße]
       [TO ganzzahlige Datengröße]
       [BY ganzzahlige Datengröße]
       [INDEX ganzzahlige Variable]

       [Anweisung ...]

   [WHILE Bedingung]

       [Anweisung ...]

    ENDDO
```

Die "Bedingung", die in der WHILE-Klausel auftritt, ist vom selben Typ von Bedingung, wie oben im Abschnitt der IF-Anweisung beschrieben wurde. Eine "ganzzahlige Variable" ist entweder ein Feld, das als INTEGER in dem Endlager definiert ist, oder es ist eine lokale DCL (Erklärung) gegenüber der Regel. Eine "ganzzahlige Datengröße" ist ein Literal, eine Setkonstante oder der Name eines Felds, das als INTEGER definiert ist. Falls nötig kann, um irgendwelche Zweideutigkeiten zu lösen, eine teilweise Qualifizierung und Tiefindizierung bei den Feldnamen gefordert werden.

Einige Beispiele von DO-Schleifen folgen:

```
DO
WHILE FICA OF RTAXCMPI <FICA_MAX_IN PARAMETERS
     Anweisung 1
     Anweisung 2

...
ENDDO

DO
     Anweisung 1
     Anweisung 2
...
WHILE TOTAL_AMT_> TOTAL_LIMIT
ENDDO

DO TO LOOP_END BY STEP INDEX COUNTER OF RXYZZYI
     Anweisung 1
     Anweisung 2
...
ENDDO

DO FROM LEVEL OF RSTATUSI
     Anweisung 1
     Anweisung 2
WHILE CODES (LEVEL) <> TERM_CODE IN VALID_CODES
...
ENDDO
```

Semantik

Bei dem ersten Typ von DO-Konstrukten -- ohne den expliziten Zählmechanismus -- werden die Anweisungen zwischen DO und ENDDO in der Reihe ihres Auftretens wiederholt, solange die Bedingung in der Klausel TRUE ist. Wenn die Bedingung FALSE wird, geht die Steuerung von der WHILE-Klausel zu der Anweisung über, die auf ENDDO folgt. Es ist anzumerken, daß die WHILE-Klausel am Beginn der Schleife, am Ende der Schleife oder irgendwo in der Mitte der Schleife auftreten kann.

Bei dem zweiten Typ von DO werden für jede fehlende Klausel (FROM, TO, BY) Ersatzwerte bereitgestellt. Falls FROM weggelassen wird, beginnt die Schleifenzählung bei 1; falls TO weggelassen wird, ist die Zählgrenze auf 32.767 festgesetzt; falls BY weggelassen wird, ist das Schleifeninkrement auf 1 festgelegt. Bemerke, daß die INDEX-Klausel nicht erscheinen muß. Zumindest eine Angabe FROM, TO, BY oder INDEX muß für das DO angegeben werden, damit es als ein indiziertes DO erkannt wird. Außerdem müssen die IN-, FROM-, TO- und BY-Werte keine ganzzahligen Konstanten sein; ganzzahlige Felder sind auch erlaubt. Letztlich kann bei einem indizierten DO zusätzlich eine WHILE-Bedingung vorgesehen sein, und sie kann überall innerhalb der Schleife auftreten.

ANHANG F: Übergabe der Steuerung zwischen Regeln

Eine Regel kann eine andere Regel oder eine Komponente aufrufen. Eine Komponente kann eine andere Komponente, aber keine Regel aufrufen.

USE-Anweisung

Syntax

Die USE-Regel-Anweisung ruft eine andere Regel auf, und die USE MODULE-Anweisung ruft eine Komponente auf. Sie sind ähnlich im Erscheinungsbild:

```
USE RULE Regel-Name        [NEST]
USE MODULE Komponenten_Name,
```

wobei Regel_Name der Name einer Regel und Komponenten_Name der Name einer Komponente ist, die dem Endlager bekannt sind.

Semantik

Bei der Ausführung von USE RULE oder USE MODULE wird die Steuerung zu der benannten Regel oder Komponente überführt. Ungleich dem Fall von Programmiersprachen sind der USE-Anweisung selbst keine der Parameter zugeordnet, die zu der aufgerufenen Regel oder Komponente zu übertragen sind.

Die Regelsprache erfordert, daß der Eingabe-View und der Ausgabe-View einer Regel in dem Endlager definiert werden, nicht innerhalb des Programms selbst. Weiterhin muß der Eingabe-View, der zu der aufgerufenen Regel übertragen wird, und der Ausgabe-View, der von ihr erhalten werden soll, in dem Endlager sowohl für die aufrufende als auch für die aufgerufene Regel verzeichnet sein.

Semantische Überprüfung der Zielumgebungen

In einem System, das die Architektur eines IBM-Mainframe eines IBM-Minicomputers, der das Stratus VOS-Betriebssystem unterstützt, und von PC-Arbeitsplätzen hat, kann nur eine Regel auf dem PC eine Regel, die zu einer abweichenden Zielumgebung gehört, aufrufen. Der Codeerzeuger wird einen Fehler anzeigen, falls man versuchen sollte, von einer Mainframe-Regel aus eine Stratus-Regel oder eine PC-Regel zu benutzen, oder falls man versuchen sollte, eine Mainframe-Regel oder eine PC-Regel von einer Regel auf dem Stratus aus zu benutzen. Es ist auch anzumerken, daß eine Regel nur eine Komponente benutzen kann, auf die in ihrer eigenen Umgebung abgezielt wird.

Semantische Überprüfungen der NEST-Klausel

Die NEST-Option zeigt an, daß alle von dieser Regel aufgerufenen Fenster und ihre aufrufenden Regeln in einen Anzeigemodus gelangen; das heißt, daß sie auf dem Schirm angezeigt werden,

der zuvor umgewandelt wurde. NEST kann nur verwendet werden, falls

- die Zielumgebung der benutzenden Regel der PC ist oder
- die Zielumgebung der benutzten Regel der PC ist.

Regeln auf anderen Maschinen können jedoch keine Regel auf dem PC aufrufen (außer indirekt durch die ASYNC-Einrichtung).

RETURN-Anweisung

Ein aufgerufenes Regelprogramm führt die RETURN-Anweisung aus, um die Steuerung zurück zu der aufrufenden Regel zu bringen. Die RETURN-Anweisung kann irgendwo in einem Programm plaziert werden. Der Regelsprachencodeerzeuger ordnet ein implizites RETURN am Ende jedes Regelprogramms an.

Übertragung von Daten zwischen Regeln

Wie in dem vorangehenden Abschnitt bemerkt wurde, können Regeln höchstens einen Input-View und einen Output-View haben, die in dem Endlager definiert sind. Dies bedenkend gelten die folgenden Beschränkungen für den Fluß beim Übertrag von Daten:

- Eine Regel kann ihren Input-View nicht ändern, falls er nicht auch ihr Output ist (d. h. INOUT).
- Eine Regel kann nicht den Output-View eines Moduls, das sie aufruft, ändern, falls er nicht das gleiche wie die Eingabe des Moduls ist.
- Eine Regel kann nicht ihren Input-View mit demjenigen der Ausgabe ihres Nachfolgers teilen. Das Teilen von Views ist erlaubt zwischen Regeln desselben Levels, wenn die Views dieselbe Verwendung haben (z. B. beide als Input).
- Output-Views von Regeln und die Input-Views von Modulen, die sie aufrufen, werden beim Aufrufen gelöscht, wie auch lokal erklärte Views.

Umwandlungsunterstützungsstrukturen

Die Fensterumwandlung (CONVERSE WINDOW) unterstützt Kommunikation zwischen Regelprogrammen in der Personal Computer-Umgebung und dem Benutzer eines Systems.

Die Anweisung hat die Form:

```
CONVERSE WINDOW WINDOW_NAME,
```

wobei WINDOW_NAME der Name ist, der der Anzeige gegeben wurde, als sie in das HPS-Endlager eingegeben wurde.

Es ist anzumerken, daß es einen Unterschied zwischen WINDOW_NAME

und Fenster-View gibt. Jeder WINDOW_NAME ist mit einem View verbunden, der sein Fenster-View genannt wird. Nicht jeder View in dem Endlager kann als ein Fenster-View verwendet werden. Ein Fenster-View kann nur einen 01 Level, 03 Level und 05 Level haben. Es ist anzumerken, daß eine Regel mehr als eine Umwandlungsanweisung aufweisen kann, aber nie mehr als einen Fensternamen. Es ist auch anzumerken, daß nur Regeln auf dem PC Umwandlungsanweisungen aufweisen können.

Asynchrone Unterstützungsstrukturen

Dieser Anhang faßt die Regelsprachenkonstrukte zusammen, die nicht angefragte Daten ausgebende Prozesse (UD) aufrufen. Nicht angefragte Daten erlauben es einer Regel, Daten zu einer anderen Regel zu senden, ohne daß die letzte Regel hiernach anfragt. Dies wird unterstützt unter Verwendung der folgenden Anweisungen:

- ASYNC ATTACH
- ASYNC DETACH
- ASYNC REFRESH
- ASYNC ROUTE

Die folgende Diskusion bezieht sich auf eine Hardwarekonfiguration mit einer Architektur, die aus einem IBM Mainframe, der das CICS-Betriebssystem unterstützt, einem Stratus Minicomputer und einer Reihe von PC-Arbeitsplätzen besteht. Alle diese Konstrukte sind derzeit beschränkt auf Regeln auf dem PC. Es gibt auch Stratusunterroutinen, um die folgenden Anweisungen auszuführen.

Beim Definieren des PC-Konstrukts werden wir die Existenz der folgenden Regelelementfälle annehmen:

- rpc1 Eine PC-Regel, die das Fenster wpc1 umwandelt
- rpc2 Eine PC-Regel, die das Fenster wpc2 umwandelt
- rpc3 Eine PC-Regel, die wpc1 auffrischt
- rpc4 Eine PC-Regel, die wpc2 auffrischt
- rst1 Eine Stratus-Regel, die eine Komponente aufruft, welche die send_message Unterroutine verwendet.

ASYNC ATTACH-ANWEISUNG

Syntax

Die ASYNC ATTACH-Anweisung hat die folgende Form:

```
ASYNC ATTACH RULE   rpc3 VIA RULE rst1
```

Dieser Befehl startet den Empfang von nicht angefragter Eingabe, die von der Stratus-Regel rst1 gesendet wird, durch die PC-Regel prc3.

Semantische Überprüfung

ATTACH kann nur verwendet werden, falls:

die Ausführungsumgebung (Endlagerattribute der angehängten Regel) der PC ist und VIA RULE STRAT ist).

ASYNC DETACH-ANWEISUNG

Syntax

Die ASYNC DETACH-Anweisung hat die folgende Form:

```
ASYNC DETACH RULE   rpc3 VIA RULE rst1
```

Dieser Befehlt führt die exakte Umkehr von ATTACH aus. Er kann nur nach einem ATTACH-Befehl verwendet werden; zu seinem Zeitpunkt unterbricht er den Empfang von nicht angefragter Eingabe, die von der Stratus-Regel rst1 gesendet wird, durch die PC-Regel rpc3,.

Semantische Überprüfung

Dieselbe wie bei ATTACH.

ASYNC ROUTE-Anweisung

Syntax

Die ASYNC ROUTE-Anweisung hat die folgende Form:

```
ASYNC ROUTE RULE    rpc3 VIA RULE rpc4
```

Dieser Befehl schaltet den Vorgang des Auffrischens eines Fensters von der PC-Regel rpc3 auf die PC-Regel rpc4 um.

Semantische Überprüfung

ROUTE kann nur verwendet werden, falls:

die Ausführungsumgebung (Endlagerattribute) von beiden Regeln der PC ist.

ASYNC REFRESH-Anweisung

Syntax

Die ASYNC REFRESH-Anweisung hat die folgende Form:

```
ASYNC REFRESH WINDOW      wpc1
     (FIELD               Feld_Name
     VIEW                 View_Name
     OCCUR                Feld_Occur
     BEEP
     FLASH)
```

wobei wpc1 ein Fenstername und Feld_Name, View_Name und Feld_Occur ein bestimmtes Feld in diesem Fenster identifizieren. Dieser Befehl leitet das automatische Aktualisieren des spezifizierten Felds für so lange ein, wie die Regel angehängt ist.

Semantik

Für das folgende sei angenommen, daß das Fenster wpc1 einen Fenster-View wv1 aufweist, der ein auftretendes Array V1 enthält; eine Listenbox, die ihrerseits V1 zugeordnet ist, umfaßt ein Feld F1. Der folgende Befehl:

```
ASYNC REFRESH  Fenster wpc1    [REFRESH-Optionen]
```

mit den REFRESH-Optionen BEEP und/oder FLASH wird ein Piep-Signal und/oder Aufleuchten der aufgefrischten Daten hervorgerufen. Eine von beiden, beide oder keine dieser Optionen können festgelegt werden.

Andere REFRESH-Schlüsselworte sind:

1.
```
FIELD     F1
VIEW      V1
OCCUR     I
```

Resultat: Bei der Listenbox V1 werden F1 oder V1 OF WV1 aufgefrischt.

2.
```
FIELD     F1
VIEW      V1
```

Resultat: Jedes Auftreten des Felds F1 bei V1 wird aufgefrischt (d. h. die Spalte).

3.
```
VIEW      V1
OCCUR     I
```

Resultat: In der Listenbox V1 wird die i'te Reihe aufgefrischt.

4.
```
VIEW      V1
```

Resultat: Die gesamte Listenbox V1 wird aufgefrischt.

Bei den Optionen 1 und 2 kann die VIEW-Klausel weggelassen werden, aber nur falls F1 von WV1 einmalig in der Regel ist.

Bei dem obigen

kann F1 der Literalname des Felds oder einer Variablen sein, die den Namen enthält (d. h. es ist entweder ein Zeichen oder ein Feld vom Typ CHAR)

kann V1 der Literalname des Views oder einer Variablen sein, die den Namen enthält (d. h. es ist entweder ein Zeichen oder ein Feld vom Typ CHAR)

ist i entweder eine Ganzzahl oder ein Feld vom Typ INTEGER.

Semantische Überprüfung

- Die WINDOW-Klausel ist obligatorisch.
- Die VIEW-Klausel, falls angegeben, muß einen Unter-View des von der WINDOW-Klausel identifizierten Views angeben.
- Die FIELD-Klausel, falls angegeben, muß einen Feldnamen spezifizieren, der in dem View enthalten ist, welcher von der VIEW-Klausel identifiziert wird.
- Falls OCCUR verwendet wird, muß der von der VIEW-Klausel identifizierte View ein auftretender View sein.
- Das OCCUR-Literal muß innerhalb von Grenzen liegen.

Richtlinien betreffend die Verwendung von ASYNC

- Eine Regel kann nicht sowohl CONVERSE-Anweisungen als auch ASYNC-Anweisungen enthalten.
- Eine Regel, die umwandelt, kann keine Regel mit einer ASYNC-Anweisung verwenden und umgekehrt.
- Einer Regel, der das im Endlager definierte Attribut ASYNC gegeben ist, kann keine USE ... NEST-Anweisung aufweisen (weil NEST impliziert, daß sie irgendwo in der Hierarchie der Regeln eine Regel mit einem CONVERSE aufruft oder in ihr selbst eine Regel mit einem CONVERSE vorhanden ist).

Jeder Fenster_Name ist einem View zugeordnet, genannt Fenster-View. Nicht jeder View in dem Endlager kann als Fenster-View verwendet werden. Ein WINDOW-View kann nur 01 Level, 03 Level und 05 Level haben. Es ist anzumerken, daß eine Regel mehr als eine CONVERSE-Anweisung, aber niemals mehr als einen Fenster-Namen haben kann. Es ist auch anzumerken, daß nur Regeln auf dem PC CONVERSE-Anweisungen haben können.

Application No.: 91 901 023.1
Applicant: Seer Technologies, Inc.
8000 Regency Parkway
US - Cary, North Carolina 27511

PATENTANSPRÜCHE

1. Verfahren zum Betreiben eines Datenprozessors zum Erzeugen und Verteilen eines Computerprogramms in einem Computersystem, das eine Mehrzahl von Hardwarekomponenten aufweist, die verbunden sind, um ein zusammengeschaltetes Computernetzwerk auszubilden, wobei Merkmale, die sich auf die Spezifikation des Computerprogramms beziehen, in Form von Datenelementen und Datenbeziehungen spezifiziert sind, **dadurch gekennzeichnet:**

a. daß der Datenprozessor als Eingabe Daten akzeptiert, die sich auf eine von dem Computerprogramm auszuführende Aufgabe beziehen, wobei die Eingabedaten gemäß einem vorausgewählten Satz von Datenelementen charakterisiert sind, wobei diese Datenelemente vorausgewählte Kategorien aufweisen,

b. daß der Datenprozessor als Eingabe in den Datenprozessor Informationen zum Verknüpfen der Datenelemente gemäß einem vorausgewählten Satz von Datenbeziehungen akzeptiert, wobei diese Datenbeziehungen vorausgewählte identifizierende Eigenschaften zwischen den Datenelementen aufweisen,

c. daß der Datenprozessor arbeitet, um die Eingabedatenelemente gemäß dem Eingabesatz der Datenbeziehungen miteinander zu verknüpfen, wobei dieser verknüpfte Satz von Datenelementen und Datenbeziehungen ein Datenmodell für die von dem Computerprogramm zu verwendenden Daten bildet,

d. daß der Datenprozessor arbeitet, um den verknüpften Satz von Datenelementen und Datenbeziehungen in einem Speicherbereich zu speichern,

e. daß der Datenprozessor als Eingabedaten Prozessorinformationen über den Ablauf des Computerprogramms in der Form von Spezifikationen für ein hierarchisches Prozeßmodell akzeptiert, wobei das hierarchische Prozeßmodell

eine Darstellung aufweist, die die Funktion des Computers in diskrete Aufgaben aufteilt, wobei die diskreten Aufgaben Beschreibungen der Funktionen vorausgewählter Abschnitte des Computerprogramms und die Verarbeitungsanforderungen für das Implementieren dieser Funktionen aufweist, wobei diese Beschreibungen zu einem vorausgewählten Satz von Prozeßelementen gruppiert sind, wobei die Prozeßelemente vorausgewählte Prozeßkategorien aufweisen,

f. daß der Datenprozessor als Eingabe Informationen zum Verknüpfen dieser Prozeßelemente gemäß einem vorausgewählten Satz von Prozeßbeziehungen akzeptiert, wobei diese Prozeßbeziehungen Beschreibungen aufweisen, um Abhängigkeiten zwischen den Prozeßelementen zu identifizieren,

g. daß der Datenprozessor arbeitet, um die Prozeßelemente gemäß dem vorausgewählten Satz von Prozeßbeziehungen zu verknüpfen, um das hierarchische Prozeßmodul zu erzeugen,

h. daß der Datenprozessor arbeitet, um das hierarchische Prozeßmodell in dem Speicherbereich zu speichern,

i. daß der Datenprozessor als Eingabe Informationen akzeptiert, die Verschaltungsdaten zu dem Computernetzwerk und Betriebsanforderungen an die Betriebsumgebungen von vorausgewählten Hardwarekomponenten spezifizieren,

j. daß der Datenprozessor arbeitet, um die Verschaltungsdaten- und Betriebsanforderungsinformationen in dem Speicherbereich zu speichern,

k. daß der Datenprozessor als Eingabe Informationen zum Verknüpfen der Verschaltungsdaten- und Betriebsanforderungsinformationen zu den Hardwarekomponeneten mit vorausgewählten Prozeßelementen, um die Betriebsumgebung für die Ausführung von vorausgewählten Prozeßelementen zu spezifizieren, akzeptiert,

l. daß der Datenprozessor arbeitet, um die Verschaltungsdaten- und Betriebsanforderungsinformationen zu den Hardwarekomponeneten mit den vorausgewählten Prozeßelementen zu verknüpfen,

m. daß der Datenprozessor als Eingabe Informationen

akzeptiert, die umgebungsunabhängige logische Konstrukte in einer ersten Programmiersprache aufweisen, wobei die logischen Konstrukte Anweisungen in der ersten Computerprogrammiersprache aufweisen, die es den Computerhardwarekomponenten ermöglichen, die in vorausgewählten entsprechenden Prozeßelementen spezifizierten diskreten Aufgaben auszuführen, wobei die erste Programmiersprache Anweisungen aufweist, die auf vorausgewählte Prozeßelemente und den verknüpften Satz von Datenelementen und Datenbeziehungen Bezug nehmen,

n. daß der Datenprozessor als Eingabe Informationen akzeptiert, die jedes logische Konstrukt mit einem entsprechenden vorausgewählten Prozeßelement in Beziehung setzen, wobei das logische Konstrukt so ausgebildet ist, daß es die Funktion, die durch sein entsprechendes Prozeßelement spezifiziert ist, ausführt,

o. daß der Datenprozessor arbeitet, um gemäß den Beziehungsinformationen jedes logische Konstrukt mit einem entsprechenden Prozeßelement zu verknüpfen,

p. daß der Datenprozessor arbeitet, um die verknüpften logischen Konstrukte und die entsprechende Eingabe, die die logischen Konstrukte mit einem Prozeßelement in Beziehung setzt, in dem Speicherbereich zu speichern,

q. daß der Datenprozessor arbeitet, um Computerprogrammmodule unter Verwendung der logischen Konstrukte, der entsprechenden vorausgewählten Prozeßelemente, anderer vorausgewählter Prozeßelemente des hierarchischen Prozeßmodells, des verknüpften Satzes von Datenelementen und Datenbeziehungen und der Eingabeinformationen, die die mit den vorausgewählten entsprechenden Prozeßelementen verknüpften Verschaltungsdaten zu den Hardwarekomponenten spezifizieren, zu erzeugen, wobei die Computerprogrammodule jeweils einen Computercode aufweisen, der von der Betriebsumgebung der Hardwarekomponente unterstützt wird, die mit dem entsprechenden vorausgewählten Prozeßelement verknüpft ist.

2. Verfahren nach Anspruch 1, weiterhin **dadurch gekennzeichnet:**

a. daß der Datenprozessor arbeitet, um den Speicherbereich abzutasten, um jedes erzeugte Codemodul den verknüpften Hardwarekomponenten zuzuordnen, und

b. daß der Datenprozessor arbeitet, um jedes erzeugte Codemodul an die verknüpften Hardwarekomponenten zu verteilen.

3. Verfahren nach Anspruch 1, weiterhin **dadurch gekennzeichnet:** daß jedes Computercodemodul ein Quellcodemodul aufweist.

4. Verfahren nach Anspruch 2, weiterhin **dadurch gekennzeichnet:** daß jedes Quellcodemodul an seine entsprechende Hardwarekomponente verteilt wird.

5. Verfahren nach Anspruch 4, weiterhin **dadurch gekennzeichnet:** daß jede Hardwarekomponente arbeitet, um ein Objektcodemodul für jedes Quellcodemodul, das an die Hardwarekomponente verteilt wird, durch Kompilieren des Quellcodemoduls zu erzeugen.

6. Verfahren nach Anspruch 3, weiterhin **dadurch gekennzeichnet:**

a. daß der Datenprozessor arbeitet, um Objektcodemodule für das Computerprogramm durch Kompilieren jedes Quellcodemoduls zu erzeugen, und

b. daß jedes Objektcodemodul an seine entsprechende Hardwarekomponente verteilt wird.

7. Verfahren nach Anspruch 3, weiterhin **dadurch gekennzeichnet,** daß der Quellcode Anweisungen in einer zweiten, höheren Computerprogrammiersprache aufweist.

8. Verfahren nach Anspruch 3, weiterhin **dadurch gekennzeichnet,** daß der Quellcode Anweisungen in einer Programmiersprache der dritten Generation aufweist, die von der Betriebsumgebung

einer vorausgewählten Hardwarekomponente unterstützt wird.

9. Verfahren nach Anspruch 7, weiterhin **dadurch gekennzeichnet,** daß der Quellcode Anweisungen in COBOL aufweist.

10. Verfahren nach Anspruch 7, weiterhin **dadurch gekennzeichnet,** daß der Quellcode Anweisungen in C aufweist.

11. Verfahren nach Anspruch 1, weiterhin **dadurch gekennzeichnet,** daß jedes Computercodemodul ein Objektcodemodul aufweist.

12. Verfahren nach Anspruch 3, weiterhin **dadurch gekennzeichnet:**

a. daß der Datenprozessor arbeitet, um jedes der erzeugten Quellcodemodule zu kompilieren und um ein entsprechendes Objektcodemodul zu erzeugen, das in der Betriebsumgebung der vorausgewählten entsprechenden Hardwarekomponente ausführbar ist, und

b. daß der Datenprozessor arbeitet, um jedes erzeugte Objektcodemodul mit einem vorausgewählten anderen Objektcodemodul gemäß der Prozeßbeziehung, die durch das hierarchische Prozeßmodell spezifiziert ist, zu verknüpfen.

13. Verfahren nach Anspruch 1, weiterhin **dadurch gekennzeichnet:**

a. daß ein Computercodekomponentenmodul in einem Datenprozessor vorgesehen ist, der konfiguriert ist, um eine vorausgewählte Funktion auszuführen, und der zum Ausführen (der Funktion) in den Betriebsumgebungen von vorausgewählten Hardwarekomponenten geeignet ist,

b. daß vorausgewählte Komponentenelemente in dem hierarchischen Prozeßmodell vorgesehen sind, die die Funktions- und Betriebsanforderungen der Objektcodekomponentenmodule beschreiben,

c. daß der Datenprozessor arbeitet, um jedes Komponentenelement mit seinem entsprechenden Computercodekomponentenmodul zu verknüpfen,

d. daß das Computercodekomponentenmodul in einem Speicher in vorausgewählten Hardwarekomponenten gespeichert wird,

e. daß die Prozeßelemente, die die Komponenten und die auf die entsprechenden Computercodekomponentenmodule Bezug nehmenden Verknüpfungsinformationen beschreiben, in dem Speicherbereich gespeichert werden,

f. daß Informationen zum Verknüpfen jedes Komponentenelements mit einem vorausgewählten Prozeßelement in dem hierarchischen Prozeßmodell als Eingabe in den Datenprozessor akzeptiert werden und

g. daß der Computer arbeitet, um jedes Komponentenelement mit einem entsprechenden vorausgewählten Prozeßelement in dem hierarchischen Prozeßmodell zu verknüpfen.

14. Verfahren nach Anspruch 13, weiterhin **dadurch gekennzeichnet,** daß der Datenprozessor als Eingabe vorausgewählte Anweisungen in der ersten Computersprache akzeptiert, die auf die Computercodekomponentenmodule Bezug nehmen.

15. Verfahren nach Anspruch 13, weiterhin **dadurch gekennzeichnet,** daß der Datenprozessor als Eingabe vorausgewählte Anweisungen in der ersten Computersprache akzeptiert, die auf die Komponentenelemente Bezug nehmen.

16. Verfahren nach Anspruch 13, weiterhin **dadurch gekennzeichnet,** daß das Computercodekomponentenmodul Objektcodeanweisungen aufweist, wobei die Objektcodeanweisungen zur Ausführung in einer Betriebsumgebung einer entsprechenden Hardwarekomponente geeignet sind.

17. Verfahren nach Anspruch 13, weiterhin **dadurch gekennzeichnet,** daß das Computercodekomponentenmodul eine graphisch gestützte Benutzerschnittstellenfunktion ausführt.

18. Verfahren nach Anspruch 13, weiterhin **dadurch gekennzeichnet,** daß das Computercodekomponentenmodul den Transfer von Daten zwischen zwei vorausgewählten Objektcodemodulen ausführt.

19. Verfahren nach Anspruch 13, weiterhin **dadurch gekennzeichnet,** daß das Computercodekomponentenmodul den Transfer von Daten zwischen zwei vorausgewählten Computercodemodulen ausführt, die vorgesehen sind, in den Betriebsumgebungen von zwei verschiedenen Hardwarekomponenten zu arbeiten.

20. Verfahren nach Anspruch 2, weiterhin **dadurch gekennzeichnet:**

a. daß der Datenprozessor als Eingabe Sicherheitszugangseingabedaten akzeptiert, die Informationen darüber aufweisen, welche Benutzer Zugang zu dem Computerprogramm und welche vorausgewählten Hardwarekomponenten Sicherheitsfreigaben zum Ausführen des Computerprogramms haben, und

b. daß der Datenprozessor arbeitet, um die Verteilung der Computercodemodule an die vorausgewählten Hardwarekomponenten gemäß der Sicherheitszugangseingabe zu beschränken.

21. Verfahren nach Anspruch 20, weiterhin **dadurch gekennzeichnet,** daß der Datenprozessor arbeitet, um den Benutzern den Zugang zu dem Computerprogramm gemäß der Sicherheitszugangseingabe zu erlauben.

22. Verfahren nach Anspruch 2, weiterhin **dadurch gekennzeichnet:**

a. daß der Datenprozessor als Eingabe Informationen zum Modifizieren der in der ersten Computerprogrammiersprache gehaltenen Anweisungen eines logischen Konstrukts akzeptiert,

b. daß der Datenprozessor arbeitet, um den Speicherbereich zu durchsuchen und um festzustellen, ob die Modifikation Änderungen der verknüpften Elemente in dem hierarchischen Prozeßmodell erforderlich macht,

c. daß der Datenprozessor arbeitet, um neue Computerprogrammodule unter Verwendung der modifizierten logischen Konstrukte, der entsprechenden vorausgewählten Prozeßelemente, anderer vorausgewählter Prozeßelemente des hierarchischen Prozeßmodells, des verknüpften Satzes von

Datenelementen und Datenbeziehungen und der Eingabeinformationen, die die mit den vorausgewählten entsprechenden Prozeßelementen verknüpften Verschaltungsdaten zu den Hardwarekomponenten spezifizieren, zu erzeugen, und

d. daß der Datenprozessor arbeitet, um die neuen Computerprogrammodule an die Betriebsumgebungen der entsprechenden vorausgewählten Hardwarekomponenten zu verteilen.

23. Verfahren nach Anspruch 22, weiterhin **dadurch gekennzeichnet,** daß die neuen Computercodemodule an die Hardwarekomponenten gemäß der Sicherheitszugangseingabe verteilt werden, wobei die Sicherheitszugangseingabe Informationen darüber aufweist, welche Benutzer Zugang zu dem Computerprogramm und welche vorausgewählten Hardwarekomponenten Sicherheitsfreigaben zum Ausführen des Computerprogramms haben.

24. Verfahren nach Anspruch 2, weiterhin **dadurch gekennzeichnet:**

a. daß der Datenprozessor als Eingabe Informationen zum Modifizieren der in vorausgewählten Prozeßelementen oder Prozeßbeziehungen zwischen zwei Prozeßelementen enthaltenen Daten akzeptiert,

b. daß der Datenprozessor arbeitet, um den Speicherbereich zu durchsuchen, um festzustellen, ob die Modifikationsinformationen Änderungen von verknüpften logischen Konstrukten erforderlich machen,

c. daß der Datenprozessor arbeitet, um Modifikationen an den verknüpften logischen Konstrukte auszuführen, die durch die Änderungen der vorausgewählten Prozeßelemente erforderlich gemacht wurden,

d. daß der Datenprozessor arbeitet, um neue Computercodemodule unter Verwendung der modifizierten logischen Konstrukte, der entsprechenden vorausgewählten Prozeßelemente, anderer vorausgewählter Prozeßelemente des hierarchischen Prozeßmodells, des verknüpften Satzes von Datenelementen und Datenbeziehungen und der Eingabe-

informationen, die die mit den vorausgewählten entsprechenden Prozeßelementen verknüpften Verschaltungsdaten zu den Hardwarekomponenten spezifizieren, zu erzeugen, und

d. daß der Datenprozessor arbeitet, um die neuen Computerprogrammodule an die Betriebsumgebungen der entsprechenden vorausgewählten Hardwarekomponenten zu verteilen.

25. Verfahren nach Anspruch 24, weiterhin **dadurch gekennzeichnet:**

a. daß der Datenprozessor arbeitet, um den Speicherbereich zu durchsuchen, um festzustellen, ob die Modifikationsinformationen Änderungen des verknüpften Satzes von Datenelementen und Datenbeziehungen erforderlich machen, und

b. daß der Datenprozessor arbeitet, um Modifikationen an dem verknüpften Satz von Datenelementen und Datenbeziehungen auszuführen, die durch die Änderungen der vorausgewählten Prozeßelemente erforderlich gemacht wurden.

26. Verfahren nach Anspruch 2, weiterhin **dadurch gekennzeichnet:**

a. daß der Datenprozessor als Eingabe Informationen zum Modifizieren der in vorausgewählten Datenelementen oder Datenbeziehungen zwischen zwei Datenelementen enthaltenen vorausgewählten Kategorien akzeptiert,

b. daß der Speicherbereich durchsucht wird, um festzustellen, ob die Modifikation Änderungen der verknüpften logischen Konstrukte oder des hierarchischen Prozeßmodells erforderlich macht,

c. daß der Datenprozessor arbeitet, um den Speicherbereich zu durchsuchen, um festzustellen, ob die Modifikationsinformationen Änderungen des verknüpften Satzes von Prozeßelementen und Prozeßbeziehungen in dem hierarchischen Prozeßmodell erforderlich machen,

d. daß der Datenprozessor arbeitet, um Modifikationen an dem verknüpften Satz von Prozeßelementen und Prozeß-

beziehungen auszuführen, die durch die Änderungen der vorausgewählten Datenelemente und Datenbeziehungen des verknüpften Satzes von Datenelementen und Datenbeziehungen erforderlich gemacht wurden,

e. daß der Datenprozessor arbeitet, um Modifikationen an den verknüpften logischen Konstrukten auszuführen, die durch die Änderungen der vorausgewählten Datenelemente und Datenbeziehungen des verknüpften Satzes von Datenelementen und Datenbeziehungen erforderlich gemacht wurden,

f. daß der Datenprozessor arbeitet, um neue Computercodemodule unter Verwendung der modifizierten logischen Konstrukte, der entsprechenden vorausgewählten Prozeßelemente, anderer vorausgewählter Prozeßelemente des hierarchischen Prozeßmodells, des verknüpften Satzes von Datenelementen und Datenbeziehungen und der Eingabeinformationen, die die mit den vorausgewählten entsprechenden Prozeßelementen verknüpften Verschaltungsdaten zu den Hardwarekomponenten spezifizieren, zu erzeugen, und

g. daß der Datenprozessor arbeitet, um die neuen Computercodemodule an die Betriebsumgebungen der entsprechenden vorausgewählten Hardwarekomponenten zu verteilen.

27. Verfahren nach Anspruch 1, weiterhin **dadurch gekennzeichnet:**

a. daß der Datenprozessor als Eingabe zusätzliche Informationen akzeptiert, die sich auf den Ablauf eines zweiten Computerprogramms beziehen,

b. daß der Datenprozessor arbeitet, um den Speicherbereich nach Gruppen von bereits in dem Speicherbereich existierenden Prozeßelementen und Prozeßbeziehungen zu durchsuchen, die dem gewünschten Ablauf des zweiten Computerprogramms ähnliche Prozeßfunktionen beschreiben,

c. daß der Teil des hierarchischen Prozeßmodells und der logischen Konstrukte und Computermodule, die dem gewünschten Ablauf des zweiten Computerprogramms ähnliche Prozeßfunktionen beschreiben, wiederverwendet werden.

28. Verfahren nach Anspruch 27, weiterhin **dadurch gekennzeichnet:**

a. daß der Datenprozessor als Eingabe zusätzliche Informationen akzeptiert, die sich auf die Funktion des zweiten Computerprogramms beziehen,

b. daß der Datenprozessor arbeitet, um die zusätzliche Information zu den entsprechenden wiederverwendeten Prozeßelementen des hierarchischen Prozeßmodells hinzuzufügen, und

c. daß der Datenprozessor arbeitet, um die Prozeßelemente, die die zusätzlichen, sich auf den Ablauf des zweiten Computerprogramms beziehenden Informationen enthalten, in dem Speicherbereich zu speichern.

29. Verfahren nach Anspruch 27, weiterhin **dadurch gekennzeichnet:**

a. daß der Datenprozessor als Eingabe zusätzliche Informationen akzeptiert, die sich auf die Funktion des zweiten Computerprogramms beziehen, wobei die Informationen gemäß neuen Prozeßelementen des vorausgewählten Satzes von Prozeßelementen spezifiziert sind,

b. daß der Datenprozessor als Eingabe Daten zum Verknüpfen jedes neuen Prozeßelements mit einem oder mehreren anderen neuen oder existierenden Prozeßelemente(n) durch eine Prozeßbeziehung des vorausgewählten Satzes von Prozeßbeziehungen akzeptiert,

c. daß der Datenprozessor arbeitet, um jedes neue Prozeßelement mit einem oder mehreren anderen neuen oder existierenden Prozeßelemente(n) durch eine Prozeßbeziehung des vorausgewählten Satzes von Prozeßbeziehungen zu verknüpfen, um ein neues hierarchisches Prozeßmodell zu erzeugen, und

d. daß das neue hierarchische Prozeßmodell in dem Speicherbereich gespeichert wird.

30. Verfahren nach Anspruch 29, weiterhin **dadurch gekennzeichnet:**

a. daß eine einer vorausgewählten Hardwarekomponente entsprechende Betriebsumgebung für jedes neue logische Konstrukt spezifiziert wird,

b. daß der Datenprozessor arbeitet, um die in der ersten Programmiersprache gehaltenen Anweisungen in jedem neuen logischen Konstrukt, den entsprechenden vorausgewählten Prozeßelementen, anderen vorausgewählten Prozeßelementen des hierarchischen Prozeßmodells, dem verknüpften Satz von Datenelementen und Datenbeziehungen und den Eingabeinformationen, die die mit den entsprechenden vorausgewählten Prozeßelementen verknüpften Verschaltungsdaten zu den Hardwarekomponenten spezifizieren, zu verwenden, und

c. daß der Datenprozessor arbeitet, um die neuen Computerprogrammodule an die Betriebsumgebungen der entsprechenden vorausgewählten Hardwarekomponenten zu verteilen.

31. Verfahren nach Anspruch 1, weiterhin **dadurch gekennzeichnet,** daß jeder der Schritte a bis q in einer der Hardwarekomponenten ausgeführt wird.

32. Verfahren nach Anspruch 30, weiterhin **dadurch gekennzeichnet,** daß jeder der Schritte a bis c in einer der Hardwarekomponenten ausgeführt wird.

33. Verfahren nach Anspruch 1, weiterhin **dadurch gekennzeichnet,** daß der Speicherbereich eine relationale Datenbank aufweist, die in einer vorausgewählten Hardwarekomponente angeordnet ist.

91 901 023.1
Seer Technologies, Inc.
8000 Regency Parkway
US - Cary, North Carolina 27511

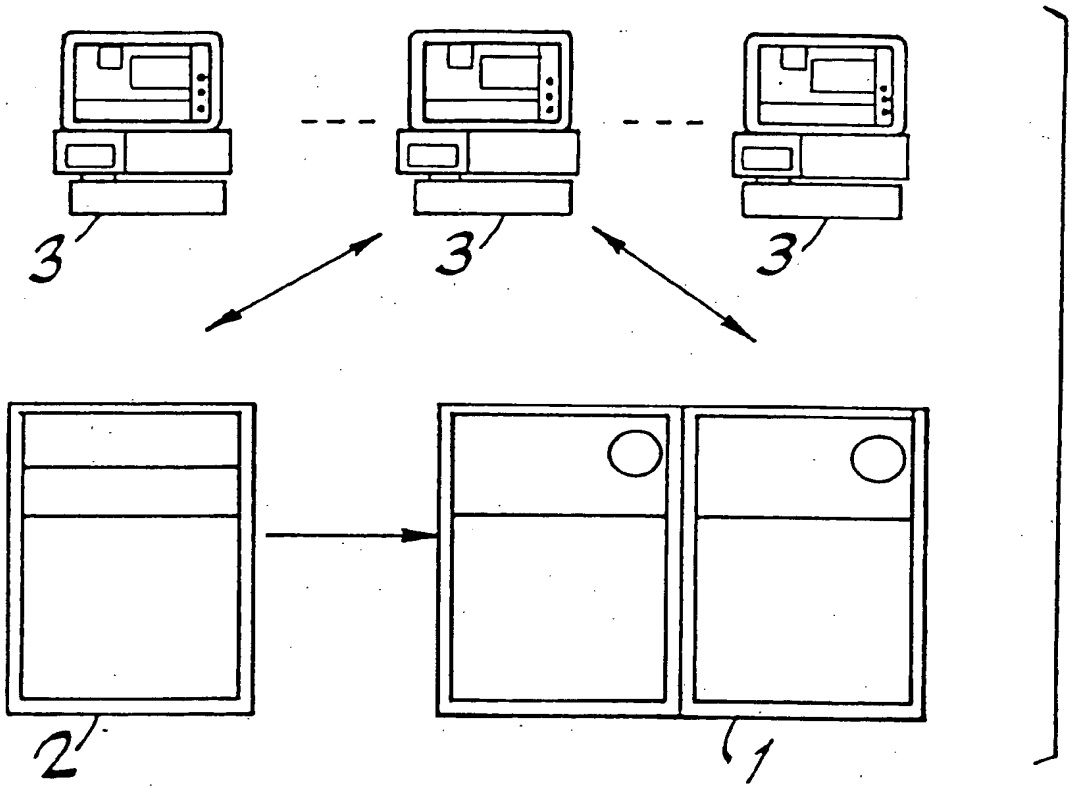

FIG. 1

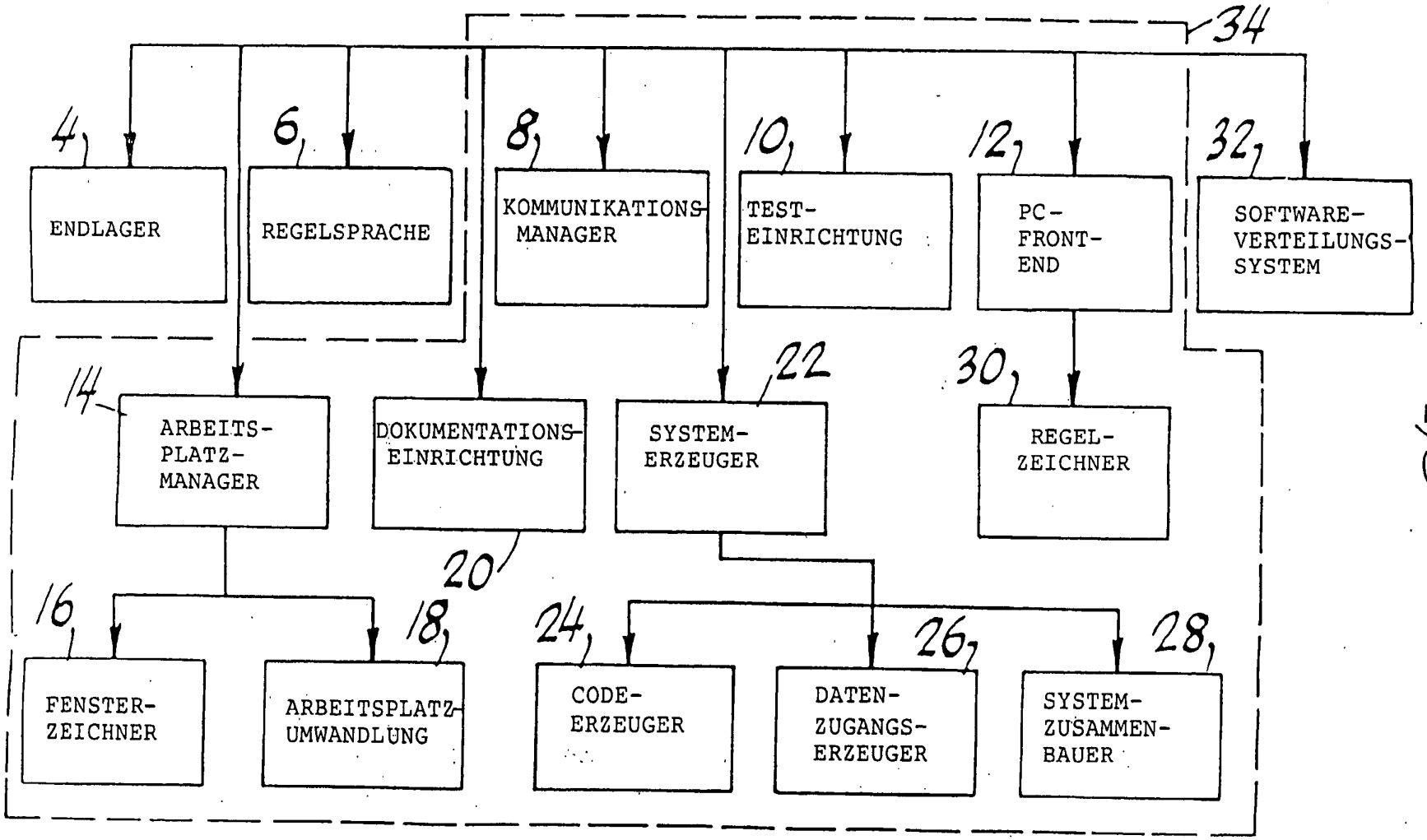
34
4
ENDLAGER
6
REGELSPRACHE
8
KOMMUNIKATIONS-MANAGER
10
TEST-EINRICHTUNG
12
PC-FRONT-END
32
SOFTWARE-VERTEILUNGS-SYSTEM
14
ARBEITS-PLATZ-MANAGER
DOKUMENTATIONS-EINRICHTUNG
22
SYSTEM-ERZEUGER
30
REGEL-ZEICHNER
20
16
FENSTER-ZEICHNER
18
ARBEITSPLATZ-UMWANDLUNG
24
CODE-ERZEUGER
26
DATEN-ZUGANGS-ERZEUGER
28
SYSTEM-ZUSAMMEN-BAUER

FIG. 2

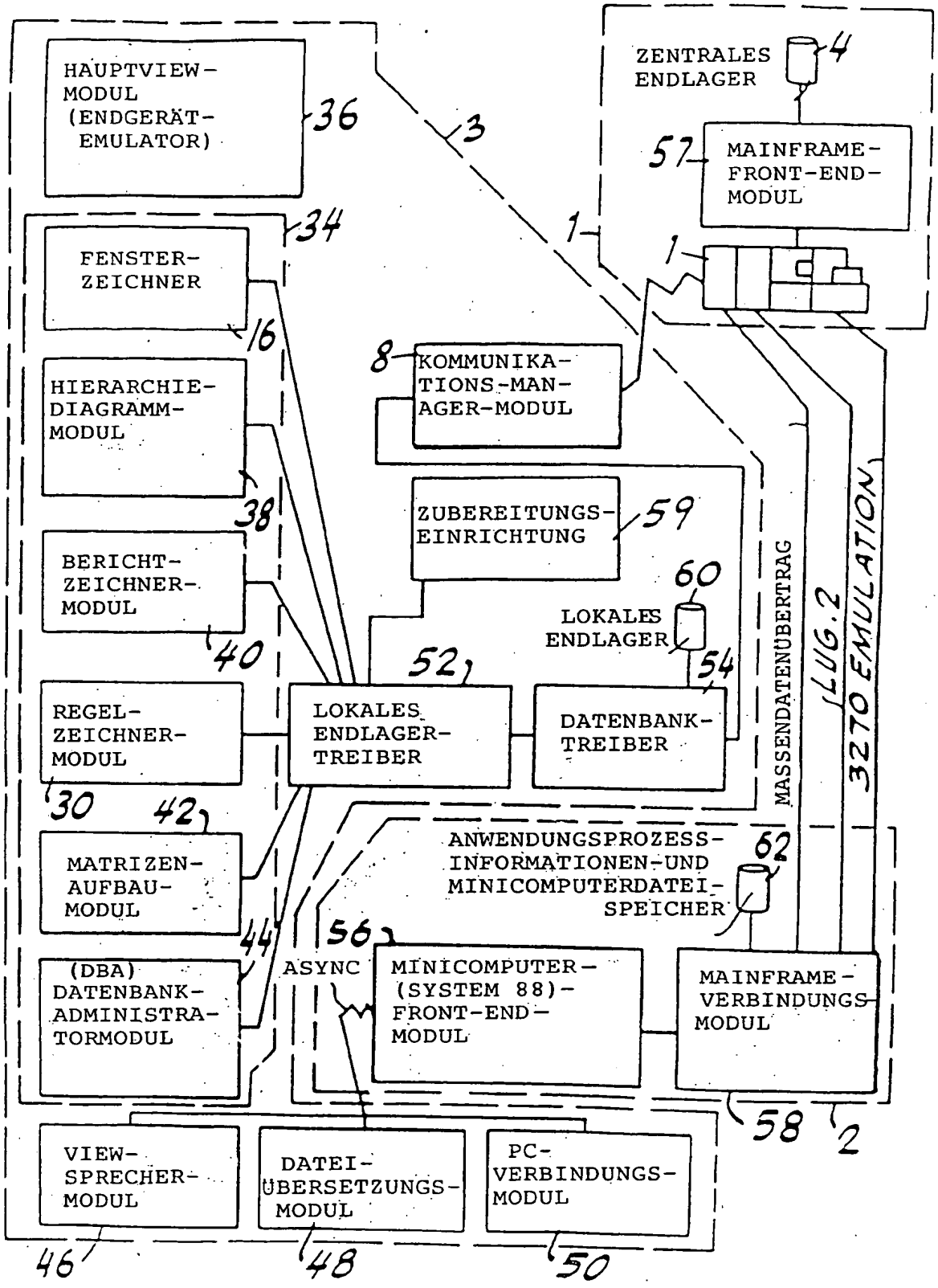
HAUPTVIEW-MODUL (ENDGERÄT-EMULATOR)
36
3
ZENTRALES ENDLAGER
4
57
MAINFRAME-FRONT-END-MODUL
1
1
34
FENSTER-ZEICHNER
16
8
KOMMUNIKATIONS-MANAGER-MODUL
HIERARCHIE-DIAGRAMM-MODUL
38
ZUBEREITUNGS-EINRICHTUNG
59
60
LOKALES ENDLAGER
54
BERICHT-ZEICHNER-MODUL
40
52
REGEL-ZEICHNER-MODUL
LOKALES ENDLAGER-TREIBER
DATENBANK-TREIBER
MASSENDATENÜBERTRAG
LUG.2
3270 EMULATION
30
42
ANWENDUNGSPROZESS-INFORMATIONEN-UND MINICOMPUTERDATEI-SPEICHER
62
MATRIZEN-AUFBAU-MODUL
44
56
ASYNC
(DBA) DATENBANK-ADMINISTRA-TORMODUL
MINICOMPUTER-(SYSTEM 88)-FRONT-END-MODUL
MAINFRAME-VERBINDUNGS-MODUL
58
2
VIEW-SPRECHER-MODUL
DATEI-ÜBERSETZUNGS-MODUL
PC-VERBINDUNGS-MODUL
46
48
50

FIG. 2A

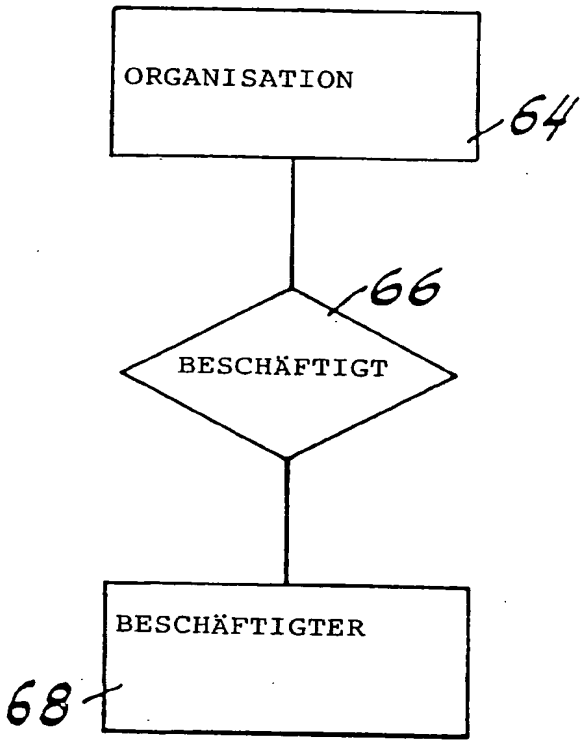

FIG. 3

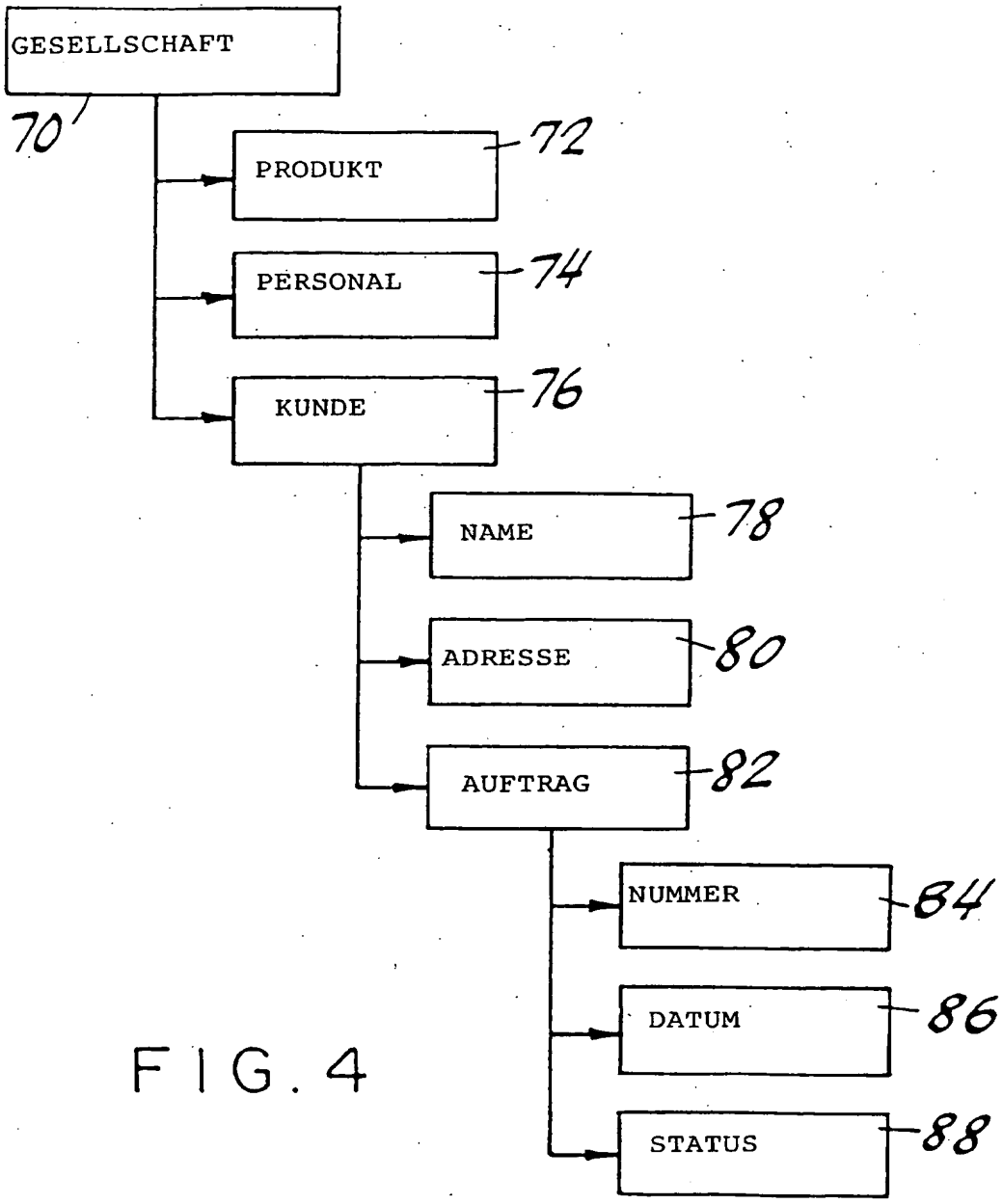
GESELLSCHAFT
70
PRODUKT
72
PERSONAL
74
KUNDE
76
NAME
78
ADRESSE
80
AUFTRAG
82
NUMMER
84
DATUM
86
STATUS
88

FIG. 4

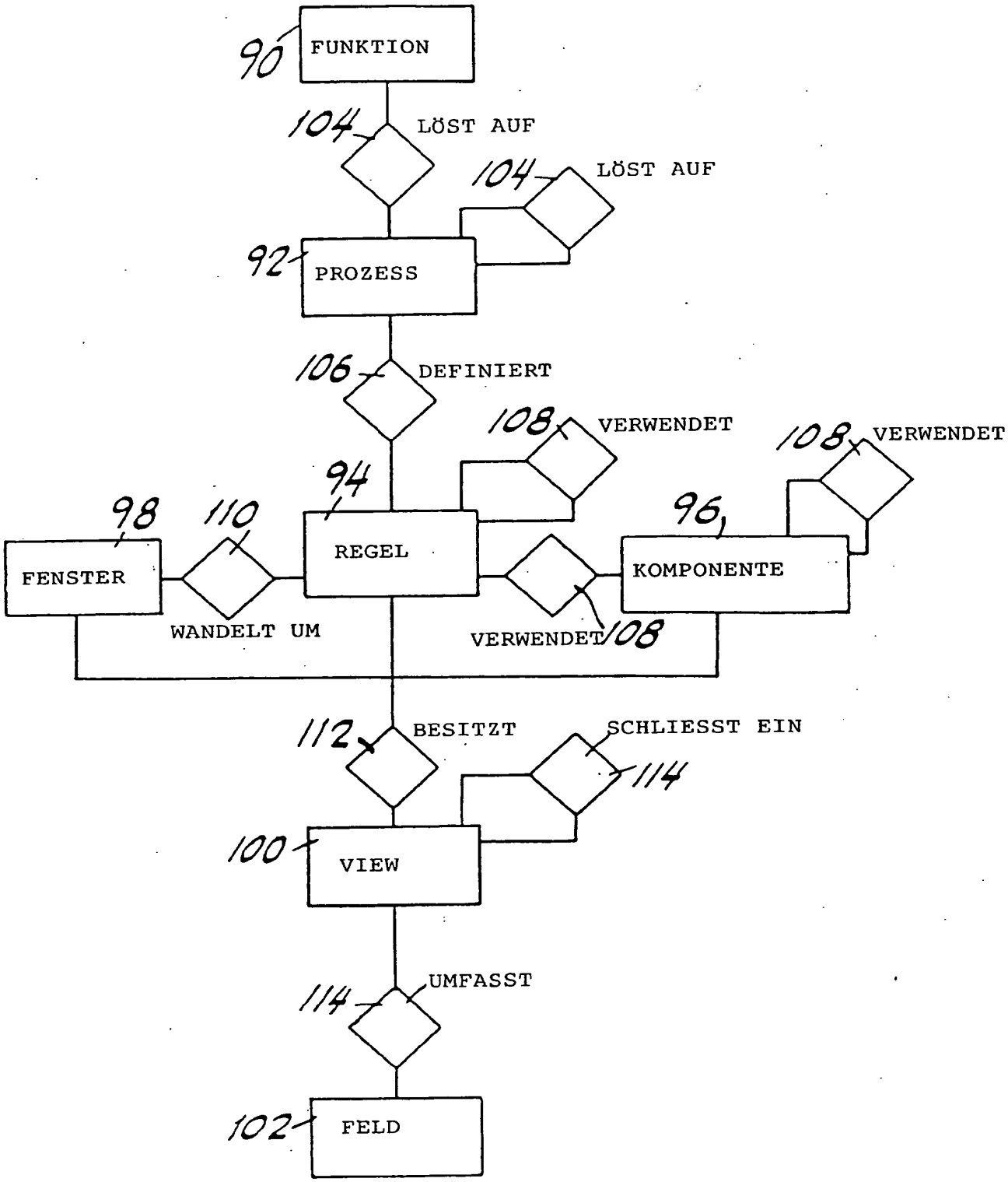
90
FUNKTION
104
LÖST AUF
104
LÖST AUF
92
PROZESS
106
DEFINIERT
108
VERWENDET
108
VERWENDET
94
98
110
96
REGEL
FENSTER
KOMPONENTE
WANDELT UM
VERWENDET
108
112
BESITZT
SCHLIESST EIN
114
100
VIEW
114
UMFASST
102
FELD

FIG. 5

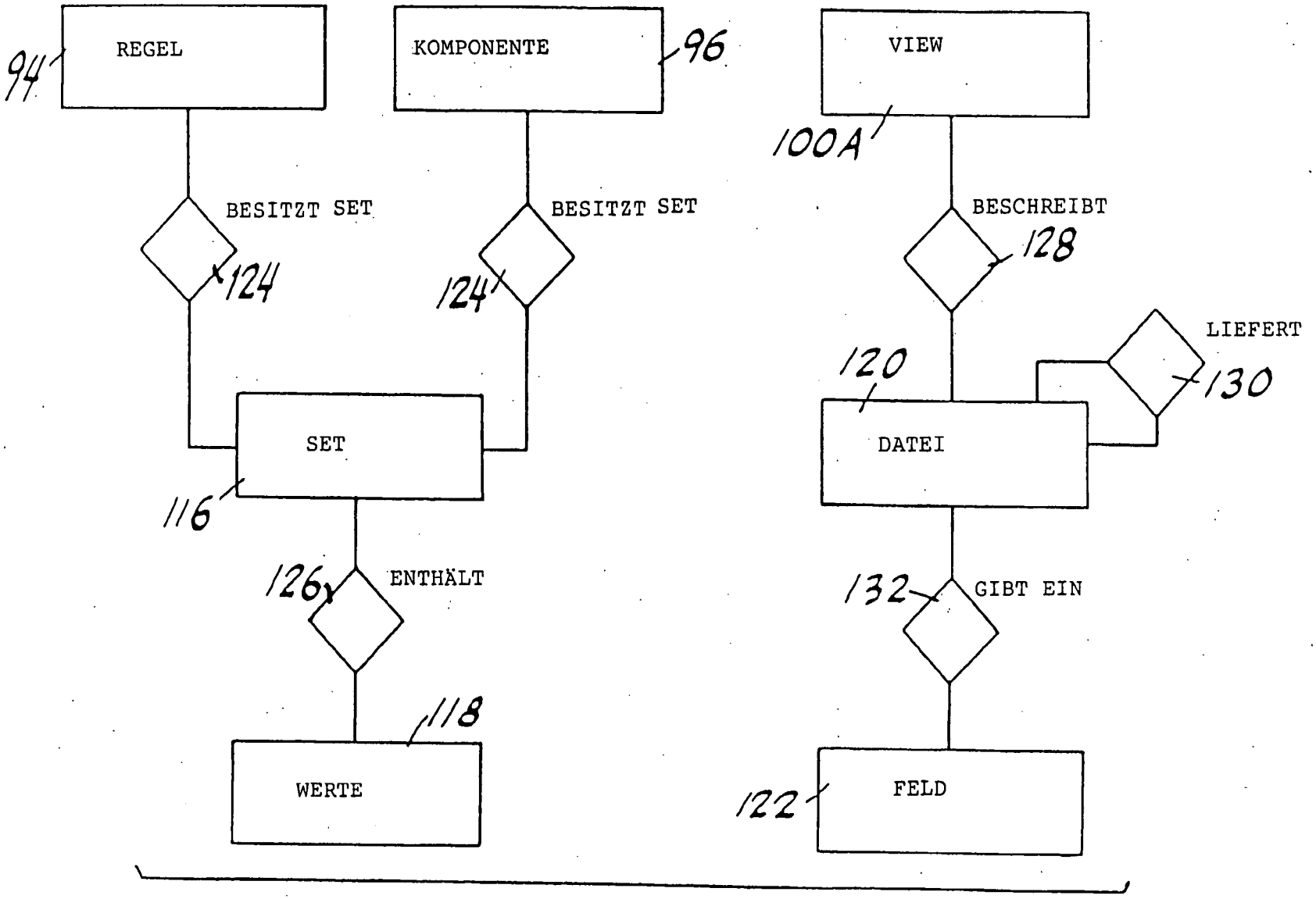
REGEL
94
KOMPONENTE
96
VIEW
100A
BESITZT SET
124
BESITZT SET
124
BESCHREIBT
128
LIEFERT
130
120
SET
116
DATEI
ENTHÄLT
126
GIBT EIN
132
118
WERTE
FELD
122

FIG. 6

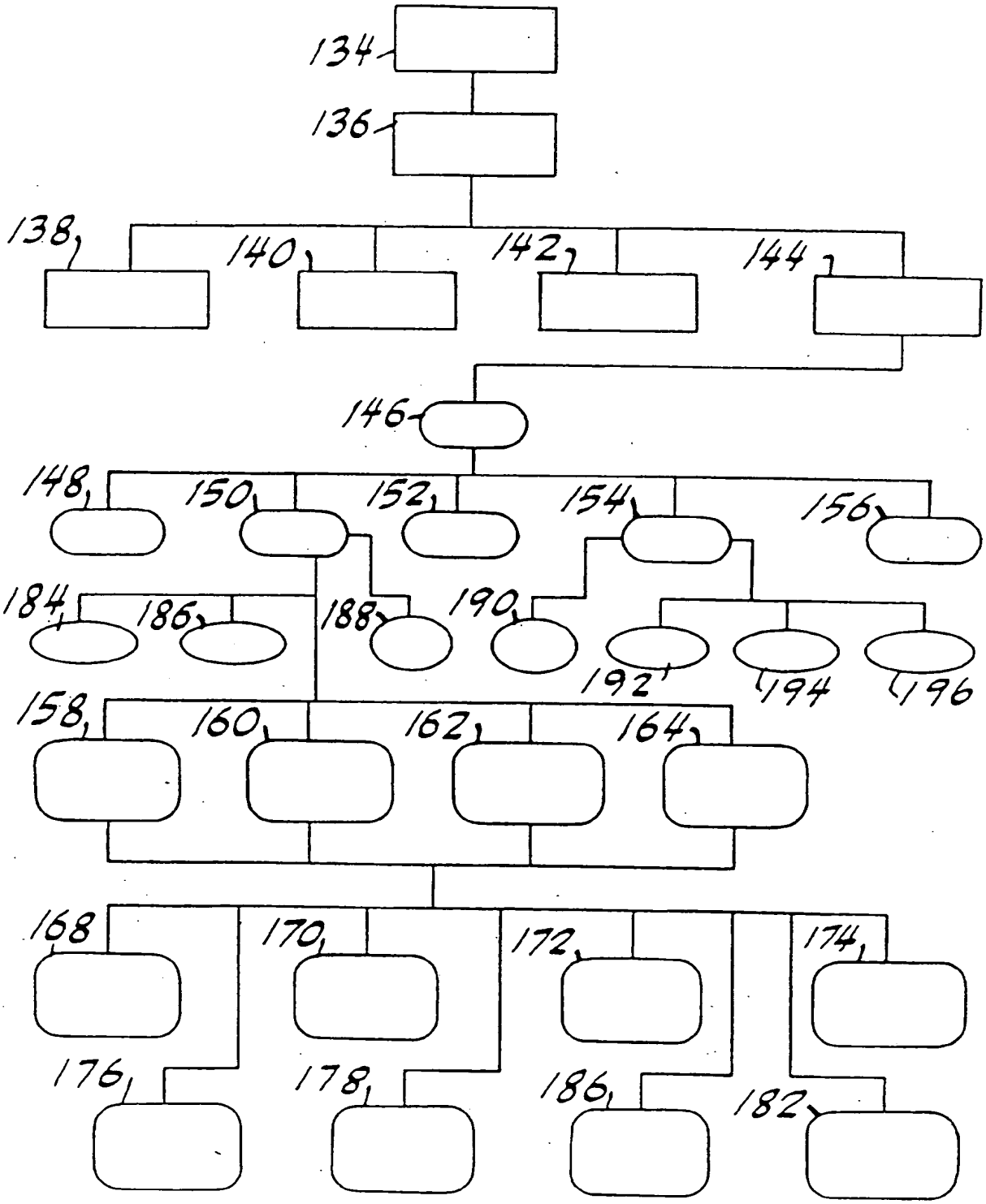
134
136
138
140
142
144
146
148
150
152
154
156
184
186
188
190
192
194
196
158
160
162
164
168
170
172
174
176
178
186
182

FIG. 7